Linux Kernel State Tracing Facility

Function Specifications

Version 01-05

Revision History

Rev	Date	Author	Description
1.00	2001.10.09	serizawa, nakamura, hatasaki	Initial version.
1.01	2001.12.14	serizawa, nakamura, hatasaki	- Add description about buffer lists (in 3., 4.2.2, 4.5.1.3, 4.5.2.4) - Add details about return values (in 4.5.1, 4.5.2) - log_cpu -> log_procssor - Event facility code is limited as LOG_KERN, and add description that "It will be changed to LOG_LKST in the future.". - Add 4.3 Device interface - Add 4.4 Initialization (from IOCTL() to initialize) - And more clarifications.
1.02	2002.03.31	serizawa, nakamura, hatasaki	- ETRC -> LKST - Add description about LKST logging daemon (in 2.1, 2.2, 4.4,). - Add description about how to insert new trace points (in 4.6). - Add descriptions about new IOCTLs and functions (in 4.7.1, 4.7.2). - Add descriptions about command interfaces (in 4.7.3). - And more clarifications.
1.03	2002.04.12	serizawa, nakamura, hatasaki	- Add some description about related projects (in 2.3) - Add description about how to insert new trace points (in 4.6). - Add descriptions about IOCTLs and functions (in 4.7.1, 4.7.2). - Add description about new command (in 4.7.3).
1.04	2002.04.26	hatasaki	- Modify description about lkstlogd command (in 4.7.3).
1.05	2002.06.27	hatasaki	- Modify description about IOCTLs and functions (in 4.7.1, 4.7.2) - Add descriptions about lkstbuf command (in 4.7.3).

Table of Contents

1. ABSTRACT **1**

2. INTRODUCTION **1**

2.1 PURPOSE OF LKST 1
2.2 DESIGN CONCEPT 2
2.3 RELATED PROJECTS 2

3. FUNCTION OVERVIEW **3**

4. DETAILED DESCRIPTION **4**

4.1 EVENTS TO BE RECORDED 4
4.1.1 Data to be recorded *4*
4.1.2 Type of events to be recorded *5*
4.2 METHOD OF RECORDING EVENTS 5
4.2.1 Mask controlling and functions to record events. *5*
4.2.2 Structure of the event buffer *6*
4.3 DEVICE INTERFACE 6
4.4 LOG OUTPUT AND FORMATTING 7
4.5 INITIALIZATION 7
4.6 INSERT NEW TRACE POINTS 8
4.7 FUNCTIONS 11
4.7.1 Device Interface *11*
4.7.2 Kernel Functions *36*
4.7.3 User Commands *55*

1. Abstract

This document describes the specifications of Linux Kernel State Tracer, which is a kernel state tracing facility for Linux systems. In the followings, Linux Kernel State Tracer is abbreviated to LKST.

2. Introduction

2.1 Purpose of LKST

Help Linux attain the reliability and availability needed by mission-critical systems.

Linux must be more reliable to be used more widely in mission-critical systems. System venders should adequately examine the feasibility of using Linux in their systems, but this is not easy for them to do because the kinds of hardware supported by Linux are still increasing rapidly and because the Linux kernel itself is evolving rapidly. LKST was therefore developed in an effort not only to provide the information kernel programmers need for debugging efficiently but also to expand availability of this debugging information, for example even in OS crashes. This will improve the quality of Linux systems by speeding the fixing of faults, debugging, and the evolution of the Linux kernel.

A kernel programmer who wants to find out what the fundamental causes of system faults are needs a log of the state transition of the kernel. Someone referring to this log can trace what happened before the faults. And a log of hardware events such as interrupts will help solve problem caused by hardware or will at least help determine whether the problem is caused by hardware or software. Such logs will make it possible for failures to be analyzed by system engineers who know little or nothing about the Linux kernel.

These state-transition and hardware-event logs will be especially important when the OS crashes and thus should be retrieved then. But the only operations available when the OS crashes are such basic ones as a simple memory dump. This means the logs should be as small as possible.

Furthermore, such logs should be collected in the system itself even when the failure occurs in a customer system. The overhead for logging thus needs to be so small overhead that it can be acceptable for customer systems already in service and so small that the logging can be made part of the standard Linux kernel.

2.2 Design concept

Obtain detailed information about kernel failures.

LKST records not only stack traces and the values of CPU registers, but also important state transitions of the kernel, modification s of important variables, and hardware events. And in addition to recording these events, it records information related to them.

Support log retrieval anytime, even when the OS crashes or enters an infinite loop.

The logs that LKST records should be preserved into files for latter fault analysis. But files of logs that are not concerned with the faults should be erased or overwritten to avoid waste of storage.

The logs also should be kept small enough that they can be retrieved by a simple memory dump. LKST should also provide a means of exporting logs, such as serial console output.

Minimize logging overhead, even in a multiprocessor system.

Because LKST is to be used in customer systems, it should require only minimal resources. And because it should be made part of the standard kernel modifications from the original should be minimized (i.e., LKST should collect information at the fewest possible points). Furthermore, LKST should avoid lock overhead in multiprocessor systems.

Supports user-extensible traps to allow customized error collection and monitoring.

Information to be collected may differ system by system. Users may need data the standard LKST features don't collect, so users should be able to customize the LKST functions called when events occur.

Conformance to the standard.

Information from LKST should conform to POSIX Draft Standard (1003.25). Also LKST should be controlled via APIs that conform to this standard.

2.3 Related projects

LKST uses Kernel Hooks as hooks in the kernel, and uses LKCD for kernel crash dump function.
The following are projects that may be of concern to those interested in LKST
(For details, refer to the respective URL).

LTT (Linux Trace Toolkit, http://www.opersys.com/LTT/index.html)
Kernel Hooks (http://oss.software.ibm.com/developerworks/opensource/linux/projects/kernelhooks/)
LKCD (Linux Kernel Crash Dump, http://oss.sgi.com/projects/lkcd/)
Linux Event Logging for Enterprise-Class Systems (http://evlog.sourceforge.net/)
dProbes (http://oss.software.ibm.com/developerworks/opensource/linux/projects/dprobes/)
Kernel Tracer in IKD (Integrated Kernel Debugging Facilities,
ftp://ftp.kernel.org/pub/linux/kernel/people/andrea/ikd/)

3. Function overview

LKST logs event information required for fault analysis at trace points in a kernel, and stores this information in a memory in order of the times at which the events occurred.

Figure 3-1 shows a block diagram of LKST

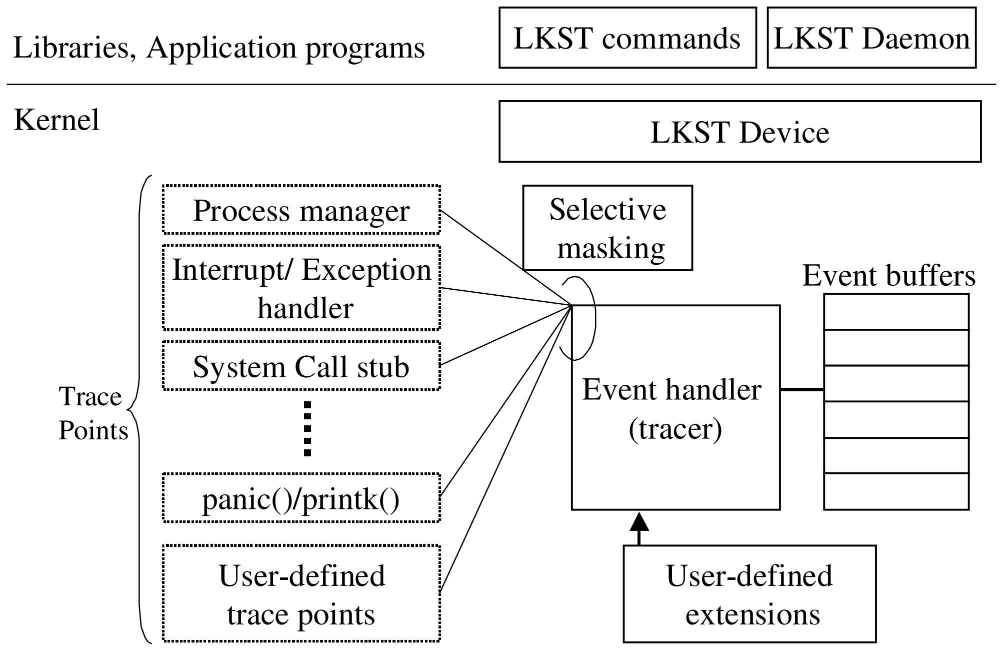

Figure 3-1. LKST block diagram.

LKST provides the following functions:

- Masking of events to be logged.
- Getting the event logs in the event buffers.
- Restoring the event logs into files. This is a function of logging daemon of LKST. When a fault occur, the daemon can preserve latest logs recorded before the fault.
- Managing the event buffers. i.e., Creating, deleting and selecting.
- Adding/Deleting the event-handler function invoked when events are logged.
- Getting LKST status.

All of the basic functions of LKST are built into a kernel, and has interfaces as a pseudo device. Therefore, a library and an application program can execute these functions through an **ioctl** system call.

LKST can get event log entries from the event buffers either by reading an event buffer (using an API that LKST provides) or by picking up event log entries from a kernel memory image obtained by existing tools etc.

4. Detailed Description

4.1 Events to be recorded

4.1.1 Data to be recorded

LKST records two kinds of data into an event log entry when an event occurred:

•Data to be recorded common to all events.

•Data to be recorded specialized by each type of event (this data is listed in a table in the Appendix).

The data for all events is listed in **Table 4-1**. These table entries conform to the POSIX Draft Standard (1003.25).

Table 4-1. Data to be recorded in all events.

Member Type	Member Name	Description
posix_log_recid_t	*Log_recid*	System-assigned ID of the event record
int	*Log_event_type*	Event identification code
uid_t	*Log_uid*	Effective user ID associated with the event
gid_t	*Log_gid*	Effective group ID associated with the event
pid_t	*Log_pid*	Process ID associated with the event
pid_t	*Log_pgrp*	Process group associated with the event
struct timespec	*Log_time*	Event time stamp
unsigned int	*Log_flags*	Bitmap of event flags
pthread_t	*Log_thread*	Thread ID associated with event
posix_log_procid_t	*log_processor*	Processor ID associated with event

LKST also records the information listed in **Table 4-2**. LKST always stores these same values into all the entries..

Table 4-2. Other information defined in POSIX (1003.25) (Fixed value).

Member Type	Member Name	Description	Value
Size_t	*log_size*	Size of the event record variable data	*sizeof(lkst_arg_t)*4*
Int	*log_format*	Format of variable data	*PXLOG_BINARY*
Posix_log_facility_t	*log_facility*	Event facility code	*LOG_KERN (*1)*
Posix_log_severity_t	*log_severity*	Event severity code	*LOG_DEBUG*

*1: This will be chainged to LOG_LKST in the future.

4.1.2 Type of events to be recorded

There are two kinds of costs for recording events: the increase in the number of dynamic steps, and the cost of maintenance due to modification of the original kernel. To minimize both costs, LKST should collect as much information as possible from the fewest invocations.

LKST limits data collection according to the following criteria:

1. Passage of paths where many control flows concentrate, such as entry points of system calls, functions that process input/output of network packets.
2. Important state transitions in the kernel, such as process states, interrupts, and exceptions.
3. Events generated by LKST itself (for maintenance).

Details are described in a table in the Appendix.

4.2 Method of recording events

4.2.1 Mask controlling and functions to record events.

LKST can control the recording of events according to whether or not the events are masked. Masked events are not recorded in the event buffer.

The mask for each type of event is controllable. In the following part of this paper, a set of masks is called a maskset. LKST can select one maskset at a time, and users need to register a maskset before it can be selected.

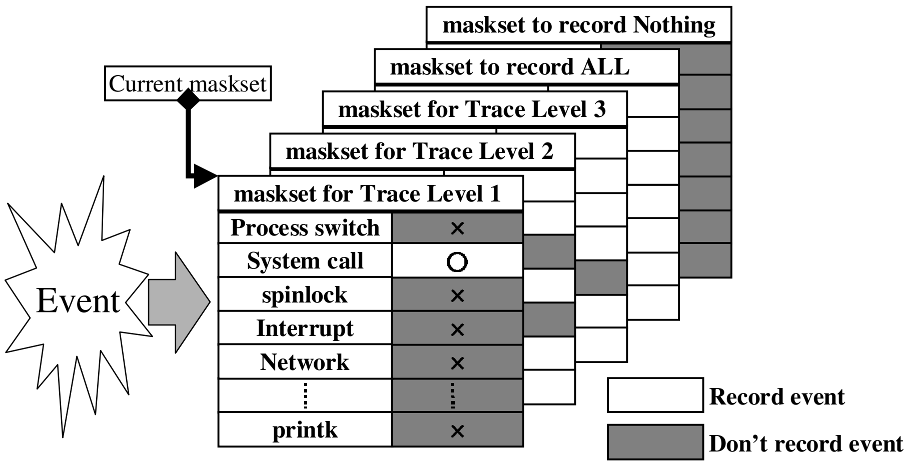

Figure 4-1. Masksets.

LKST has the following three masksets by default: (1) a maskset to record no event, (2) a maskset to record all events, (3) a maskset to record events defined before kernel compilation (not shown in Figure 4-1).

Users suspend the recording of events by selecting the first type of these maskset and resume recording by selecting any other maskset.

Events are recorded in the event buffer by a function called an “event handler,” and an event handler can be defined for each type of event. Therefore, actual contents of a maskset consists are pairs of an ID for type of event and an ID of event-handler. In the followings, this pair called “maskset entry”.
Furthermore, users can register a user-defined function as an optional event handler. Using this mechanism, for example, users can register afunction that compresses events in the event buffer.

4.2.2 Structure of the event buffer

The event buffer is actually a set of lists of circular buffers. To avoid lock overhead in multiprocessor systems, LKST has different list for each CPU. The buffer to record can be changed by using IOCTL or a kernel function. Therefore, while execution of interested command, use different buffers for recording in order to avoid buffer overrun. Users can retrieve any buffer, and can free unused buffers. Furthermore, users can create new buffer even when LKST is active. The buffer creator will add the new buffer on the tail of the list of buffers specified by a user.

In the lists, each buffer has two pointers pointing next and previous buffer. Said IOCTL and the kernel function shift current buffer to the buffer pointed by the next pointer.

There are two limitations of the buffer. The first is that the size of a buffer will be limited as power of two. The second is that LKST will not use all of entries. Several entries in the head of buffers preserved for events recorded while buffer shifting.

On overrun, LKST generates an “Overrun event”[1] by itself. As mentioned above, each event can correspond to an independent event handler, and users can customize the way LKST behaves when an overrun occurs.

The buffer which has ID=0 is for special purpose. LKST uses it to record events occured in initialization procedure, or events of LKST internal error. This buffer is not in any lists, nor deletable.

4.3 Device interface

LKST has a character device for user interface. The major number of the device is displayed in /proc/devices on the entry of "lkst". And the minor number is 0.

This device accepts only open(), close(), ioctl() and read(). Only the logging daemon of LKST issue read().

[1] The event type of this event is "LKST_BUFF_OVFLOW".

4.4 Log output and formatting

To get logs, a user can choose several methods for the purpose. Purposes and methods are described in **Table 4-3**.

Table 4-3. Purposes and methods of getting logs.

Methods	**Purpose**				**Description**
	OS failure	**Service failure**	**Kernel debugging**	**Performance evaluation, etc**	
Magic SysRQ	○	×	◎	△	LKST outputs logs to a console when SysRQ key is pushed. It can be used on Kernel failure but size will be limited.
LKST command	×	○	△	◎	"lkstbuf read" command. It is easy to use, and suits to use in a shell scripts.
with LKCD	◎	×	○	×	LKCD patched by LKST can extruct LKST buffers from a crash dump.
Logging daemon	×	◎	△	△	This daemon can preserve latest logs recorded before the fault, and also can limit the sum of files used as output.

4.5 Initialization

LKST is automatically initialized. If LKST is configured as a kernel module, initializer of LKST is called from module initializer. Otherwise, it is called kernel initializer in linux/init/main.c.

Initializer of LKST executes following processes to start LKST:

- Allocate event-buffers
- Allocate the memory area needed for the control area
- Register special masksets
- Register event-handler-functions for default use
- set current maskset as LKST_MASKSET_ID_RDEFAULT; i.e., the initializer starts recording of events.

The special maskets are the three kinds of masksets described below:

#define LKST_MASKSET_ID_RNOTHING 0

Maskset for recording no event

#define LKST_MASKSET_ID_RALL 1

Maskset for recording all events

#define LKST_MASKSET_ID_RDEFAULT 2

Maskset for recording events defined before kernel compilation

Furthermore, Maskset ID=3 is created by a rc script for LKST, and used as a default maskset.

4.6 Insert new trace points

LKST can trace events of user's programs or modules by inserting trace points.
Procedure to insert trace points is described blow:

(1) Add definition of a new event type to lkst_events.h (*) as follows:

```
LKST_ETYPE_DEF(event-ID, priority, hook-type, event-name, description)

  event-ID     --- Value of the event type(0x000..0xfff)
                   0x000-0x0ff        preset by LKST for kernel events.
                   0x100-0x1ff        for user use
                   0x200-0xeff        reserved
                   0xf00-0xfff        for LKST internal use
  priority    --- Priority of the event type(0x00..0xff)
  hook-type    --- Type of hook header for Kernel Hooks
                        Specify either the following according to the insertion
                        location of the HOOK macro.

                        - NORMAL: If you insert HOOK macro in the kernel,
                                  use this type.

                        - MODULE: If you insert HOOK macro in the module,
                                  use this type.

                        - INLINE: If you insert HOOK macro in the in-line
                                  function of the kernel, use this type.
                                  If you insert the same HOOK macro,
                                  in the two or more places, use this.

                       N  OTE: If you insert HOOK macro in the in-line
                              function of the module, use MODULE type.
  event-name   --- Mnemonic of the event type
  description --- Description of the event type

 (example)
      LKST_ETYPE_DEF(0x100, 0x0A, NORMAL, NEW_EVENT, NEW_EVENT)
```

(*) "/include/linux/lkst_events.h" in LKST kernel source.

(2) Insert hook macro where users want to trace.

Add following sentence to the file to be added the trace point.

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

- LKST_HOOK_INLINE(event-name, argument1, argument2, argument3, argument4)
  The case of inserting HOOK macro in the in-line function or the
  macro function. If you insert the same HOOK macros in the two or more
  places, use this.

- LKST_HOOK(event-name, argument1, argument2, argument3, argument4)
  The case except the above-mentioned.

* event-name	: It should be the same as what defined by the LKST_ETYPE_DEF macro.
* argument1..4	: 64Byre long data aquired at the trace point.

If user want to get 32bit data, use LKST_ARG32() macro.

```
LKST_ARG32(high, low)      high: upper 32bit / low: lower 32bit
```

And If user want to get pointer data, use LKST_ARGP() macro.

```
LKST_ARGP(pointer)
(LKST_ARGP() absorbs difference between architectures (64bit/32bit)))
```

(Example)

```
LKST_HOOK(0x100, LKST_ARG32(data1, 0), LKST_ARG32(data2, data3),
                    LKST_ARG32(data4, 0), LKST_ARGP(ptr1))
```

<Insert trace points to module>

When the trace point is in the module, the macro for declaration of HOOK header is put in it. And the macro for the initialization and termination of HOOK is inserted respectively in the function of the module-initialization and module-termination.

Declaration macro for HOOK header

Insert the following macro in the module. (Do not insert within any functions in the module)

```
LKST_HOOK_DECLARE(event-name)              -- Declaration for LKST_HOOK
LKST_HOOK_DECLARE_INLINE(event-name) -- Declaration for LKST_HOOK_INLINE
```

Initialization macro

Insert the following macro in initialization function of the module.

```
LKST_HOOK_INIT_MODULE(event-name, ret-variable)
```

Termination macro

Insert the following macro in cleanup function of the module.

```
LKST_HOOK_TERMINATE_MODULE(event-name, ret-variable)
```

* event-name : It should be the same as what defined by the LKST_ETYPE_DEF macro.

* ret-valiable : variable which receives return value.

(Example)

```
LKST_HOOK_DECLARE(NEW_EVENT);
static int __init testmod_init(void)
{
        int ret;
        LKST_HOOK_INIT_MODULE(NEW_EVENT, ret);

        return ret;
}
static void __exit testmod_exit(void)
{
        int ret;
        LKST_HOOK_TERMINATE_MODULE(NEW_EVENT, ret);
}
module_init(testmod_init);
module_exit(testmod_exit);
```

4.7 FUNCTIONS

4.7.1 Device Interface

All IOCTL commands must be called by the superuser.

4.7.1.1 Controling LKST Status

4.7.1.11. ioctl(LKST_IOC_TRC_STATUS)

<FUNCTION>

Return a current status of LKST.

<SYNOPSIS>

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

int ioctl(int fd, int request, struct lkst_trc_status *trc_status)
```

<ARGUMENTS>

fd	file descriptor(Return value opening LKST device.)
request	value "LKST_IOC_TRC_STATUS"
trc_status	address of an **lkst_trc_status** structure object

```
struct lkst_trc_status{
        lkst_maskset_id    current_maskset_id;      /* currently selected maskset ID */
        lkst_buffer_id     current_buffer_id[LKST_CPU_MAX];
                                                    /* currently selected buffer ID */
        int     maskset_num;                        /* total number of registered masksets */
        int     evhandler_num;                      /* total number of registered event handlers */
        int     buffer_num;                         /* total number of registered event buffers */
};
```

<RETURN VALUE>

0	success
EINVAL	Argument **trc_status** is invalid.
EPERM	Was called by someone other than the superuser..

<DESCRIPTION>
Return a current status of LKST to a user -specified area.
On success, this IOCTL stores the current status of LKST in an area to which the argument **trc_status** points as a structure **lkst_trc_status** type, and returns 0. A currently selected maskset ID is stored in the member **current_maskset_id** of the **lkst_trc_status** structure object. Currently selected kernel-event buffer IDs for all the CPUs are stored in the member **current_buffer_id**. The total number of registered masksets is stored in the member **maskset_num**. The total number of event handlers is stored in the member **evhandler_num**, and the total number of kernel-event buffers is stored in the member **buffer_num**.
On error, this IOCTL returns a nonzero value described above, and the values of the argument are not assured.

<REFERENCES>
ioctl(LKST_IOC_TRC_START), ioctl(LKST_IOC_TRC_STOP),
ioctl(LKST_IOC_MASKSET_LIST), ioctl(LKST_IOC_EVHANDLER_LIST),
ioctl(LKST_IOC_BUFFER_LIST)

4.7.1.12. ioctl(LKST_IOC_TRC_START)

<FUNCTION>
Start LKST event tracing.

<SYNOPSIS>
#include <linux/lkst.h>

int ioctl(int fd, int request)

<ARGUMENTS>

fd	file descriptor(Return value opening LKST device.)
request	value "LKST_IOC_TRC_START"

<RETURN VALUE>

0	success
EPERM	Was called by someone other than the superuser.

<DESCRIPTION>
Start LKST event tracing.
On success, this IOCTL changes currently selected maskset to the maskset ID which has been saved by ioctl(**LKST_IOC_TRC_STOP**), and returns 0. If the saved maskset has been deleted, currently selected maskset is changed to **LKST_MASKSET_ID_RDEFAULT**.
On error, this IOCTL returns a nonzero value described above.

<REFERENCES>
ioctl(LKST_IOC_TRC_STATUS), ioctl(LKST_IOC_TRC_STOP),

4.7.1.13. ioctl(LKST_IOC_TRC_STOP)

<FUNCTION>

Stop LKST event tracing.

<SYNOPSIS>

#include <linux/lkst.h>

int ioctl(int fd, int request)

<ARGUMENTS>

fd	file descriptor(Return value opening LKST device.)
request	value “LKST_IOC_TRC_STOP”

<RETURN VALUE>

0	success
EPERM	Was called by someone other than the superuser.

<DESCRIPTION>

Stop LKST event tracing.

On success, this IOCTL changes currently selected maskset to **LKST_MASKSET_ID_RNOTHING**, and saves previously selected maskset ID, and returns 0. The saved maskset ID is used by ioctl(**LKST_IOC_TRC_START**).

On error, this IOCTL returns a nonzero value described above.

<REFERENCES>

ioctl(LKST_IOC_TRC_STATUS), ioctl(LKST_IOC_TRC_START),

4.7.1.2 Maskset Control

4.7.1.21. ioctl(LKST_IOC_MASKSET_READ)

<FUNCTION>

Read contents of maskset.

<SYNOPSIS>

```
#include <linux/lkst.h>
#include <linux/lkst_maskset.h>
#include <linux/lkst_evhandler.h>

int ioctl(int fd, int request, struct lkst_maskset_param *maskset_param)
```

<ARGUMENTS>

fd	file descriptor(Return value opening LKST device.)
request	value 'LKST_IOC_MASKSET_READ"
maskset_param	address of an **lkst_maskset_param** structure object

```
struct lkst_maskset_param {
        lkst_maskset_id     id;                    /* maskset ID */
        size_t     maskset_size;                   /* maskset size*/
        struct lkst_maskset_body     *maskset      /* address of a maskset contents returned area */
};

struct lkst_maskset_body {
        char      name[LKST_MASKSET_NAME_LEN];     /* maskset name */
        lkst_maskset_table_len      len;           /* total number of maskset entries */
        struct lkst_maskset_entry   entry[LKST_MASKSET_TABLE_LEN_MAX];
                                                   /* maskset entry */
}

struct lkst_maskset_entry {
        int event_type;                            /* corresponding type of event */
        lkst_evhandler_id     id;                  /* event handler ID */
}
```

<RETURN VALUE>

0	success
ENOMEM	Kernel cannot allocate memory area to be used by this IOCTL.
EINVAL	Argument **maskset** or **maskset_size** is invalid.
and/or	Specified maskset ID (**id**) is invalid.
and/or	Specified maskset does not exist.
EPERM	Was called by someone other than the superuser.

<EXPLANATION>

Return contents of specified maskset.
A maskset to be read is specified by the member **id** of an **lkst_maskset_param** structure object to which the argument **maskset_param** points. If **id** specifies **LKST_MASKSET_ID_VOID**, currently selected maskset is specidifed automatically. The member **maskset_size** specifies the size of the area to which the contents of maskset are returned (*), and the member **maskset** specifies the virtual address of the area.
On success, this IOCTL stores the contents of the specified maskset in the area that the member **maskset** points to as a structure **lkst_maskset_body** type, and returns 0. The name of the maskset is stored to the member **name** as a null-terminated string. A type of event and a corresponding event handler ID are stored in the member **entry** as a structure **lkst_maskset_entry** type, and the total number of maskset entries is stored as the member **len**. The member **event_type** and **id** of the **lkst_maskset_entry** structure object respectively specify the type of event and the event handler ID.
On error, this IOCTL returns a nonzero value described above. In this case, the values of the argument are not assured.
(*) To get **maskset_size**, use **LKST_MASKSET_SIZE**([number of maskset entries]) macro**.**

<REFERENCES>

ioctl(LKST_IOC_MASKSET_WRITE), ioctl(LKST_IOC_MASKSET_SET),
ioctl(LKST_IOC_MASKSET_LIST), ioctl(LKST_IOC_MASKSET_DELETE)

4.7.1.22. ioctl(LKST_IOC_MASKSET_WRITE)

<FUNCTION>

Register a new maskset

<SYNOPSIS>

```
#include <linux/lkst.h>
#include <linux/lkst_maskset.h>
#include <linux/lkst_evhandler.h>

int ioctl(int fd, int request, struct lkst_maskset_param *maskset_param)
```

<ARGUMENTS>

fd	file descriptor(Return value opening LKST device.)
request	value ‘LKST_IOC_MASKSET_ WRITE”
maskset_param	address of an **lkst_maskset_param** structure object

```
struct lkst_maskset_param {
        lkst_maskset_id     id;                       /* maskset ID */
        size_t    maskset_size;                       /* maskset size*/
        struct lkst_maskset_body     *maskset         /* address of a maskset stored area */
};

struct lkst_maskset_body {
        char      name[LKST_MASKSET_NAME_LEN];        /* maskset name */
        lkst_maskset_table_len      len;              /* total number of maskset entries*/
        struct lkst_maskset_entry   entry[LKST_MASKSET_TABLE_LEN_MAX];
                                                      /* maskset entry */
}

struct lkst_maskset_entry {
        int event_type;                               /* corresponding type of event */
        lkst_evhandler_id     id;                     /* event handler ID */
}
```

<RETURN VALUE>

0	success
ENOMEM	Kernel cannot allocate memory area to be used by this IOCTL.
and/or	Memory area for the new maskset exceeds LKST available area.
EINVAL	Argument **maskset** or **maskset_size** is invalid.
and/or	Specified maskset ID is invalid.
and/or	Specified **event_type** is invalid.
and/or	Specified event-handler ID is invalid.
and/or	Specified event-handler does not exist.
and/or	Specify to record lock events with waking daemon process up .
EBUSY	Specified maskset is collapsed (Overwrite case).
and/or	No available Maskset ID .
EPERM	Was called by someone other than the superuser.

<DESCRIPTION>

Register a new maskset.

This IOCTL loads a new maskset specified by the member **maskset** of an **lkst_maskset_param** structure object to which the argument **maskset_param** points. The size of the maskset is specified by the member **maskset_size** (*). The member **id** of the structure specifies the ID of new maskset. If **id** specifies **LKST_MASKSET_ID_VOID**, unused ID is allocated automatically. The allocated ID is sored in the member **id**. Users store contents of the new maskset in the area that the member **maskset** points to as a structure **lkst_maskset_body** type. The maskset name is stored as the member **name** of the **lkst_maskset_body** structure object, as a null-terminated string. A type of event and a corresponding event handler ID are stored in the member **entry** as a structure **lkst_maskset_entry** type, and the total number of maskset entries is stored as the member **len**. The member **event_type** and **id** of the **lkst_maskset_entry** structure object respectively specify a type of event and an event handler ID.

On success, this IOCTL registers the member **maskset** of the **maskset_param** structure object that a user prepared, and this IOCTL returns 0.

On error (e.g., the specified event handler does not exist), this IOCTL returns a nonzero value described above. In this case, the values of the argument are not assured.

(*) To get **maskset_size**, use **LKST_MASKSET_SIZE(**[number of maskset entries]**)** macro.

<Attention> Users do not allow recording lock events(event_type is 0x080 – 0x08F) while waking daemon process up(specify event-handler **LKST_EVHANDLER_ID_BUFFER_SHIFT_DW** as event handler of **LKST_ETYPE_LKST_BUFF_OVFLOW** event).

<REFERENCES>

ioctl(LKST_IOC_MASKSET_READ), ioctl(LKST_IOC_MASKSET_SET),
ioctl(LKST_IOC_MASKSET_LIST), ioctl(LKST_IOC_MASKSET_DELETE)

4.7.1.23. ioctl(LKST_IOC_MASKSET_SET)

<FUNCTION>

Switch a currently selected maskset

<SYNOPSIS>

```
#include <linux/lkst.h>
#include <linux/lkst_maskset.h>

int ioctl(int fd, int request, lkst_maskset_id id)
```

<ARGUMENTS>

fd	file descriptor(Return value opening LKST device.)
request	value 'LKST_IOC_MASKSET_ SET"
id	maskset ID

<RETURN VALUE>

0	success
EINVAL	Specified new maskset ID is invalid.
and/or	Specified new maskset does not exist.
EBUSY	Currently selected maskset is not initialized.
and/or	Try to change maskset while LKST is stopped.
EPERM	Was called by someone other than the superuser.

<DESCRIPTION>
Switch the currently selected maskset to a specified maskset.
On success, this IOCTL switches the current maskset to a maskset that the argument **id** specifies and returns 0.
On error (e.g., the specified maskset does not exist), this IOCTL returns a nonzero value described above.

<REFERENCES>
ioctl(LKST_IOC_MASKSET_READ), ioctl(LKST_IOC_MASKSET_WRITE),
ioctl(LKST_IOC_MASKSET_LIST), ioctl(LKST_IOC_MASKSET_DELETE),
ioctl(LKST_IOC_TRC_STOP)

4.7.1.24. ioctl(LKST_IOC_MASKSET_LIST)

<FUNCTION>
Return a list of registered masksets

<SYNOPSIS>

```
#include <linux/lkst.h>
#include <linux/lkst_maskset.h>

int ioctl(int fd, int request, struct lkst_maskset_listparam *maskset_listparam)
```

<ARGUMENTS>

fd	file descriptor(Return value opening LKST device.)
request	value 'LKST_IOC_MASKSET _LIST"
maskset_listparam	address of an **lkst_maskset_listparam** structure object

```
struct lkst_maskset_listparam {
        lkst_maskset_id                 current_id;             /* current maskset ID */
        size_t      listent_size;                               /* size of the listent */
        struct lkst_maskset_listent     *listent                /* area to store the list of masksets */
};
```

```
struct lkst_maskset_listent {
        lkst_maskset_id        id;                              /* maskset ID */
        char      name[LKST_MASKSET_NAME_LEN];    /* maskset name */
        lkst_maskset_table_len       len;                       /* total number of maskset entries */
}
```

<RETURN VALUE>

0	success
ENOMEM	Kernel cannot allocate memory area to be used by this IOCTL
EINVAL	Argument **listent** and/or **listent_size** is invalid.
EPERM	Was called by someone other than the superuser.

<DESCRIPTION>

Return a list of IDs, names, and total number of entries of registered masksets.

The argument **maskset_listparam** is the address of an **lkst_maskset_listparam** structure object, the member **listent** specifies the area to which result is returned as a list of structure **lkst_maskset_listent** type. In the list, this IOCTL stores entries in ascending order of maskset ID. The member **listent_size** specifies the size of the area (*). If **listent_size** is smaller than actual size of the list, this IOCTL stores the list up to the size of **listent_size**. Each maskset ID is stored as the member **id** of the **lkst_maskset_listent** structure object, name of each maskset is stored as the member **name**, and the total number of maskset entries is stored as the member **len**.

On success, this IOCTL stores the list into **listent**, and stores currently selected maskset ID into **current_id**, and returns 0.

On error, This IOCTL returns a nonzero value described above. In this case, the values of the argument are not assured.

(*) To get **listent_size**, use **ioctl(LKST_IOC_TRC_STATUS)** for getting total number of masksets and then use **LKST_MASKSET_LISTENT_SIZE**([number of masksets]) macro**.**

<REFERENCES>

ioctl(LKST_IOC_MASKSET_READ), ioctl(LKST_IOC_MASKSET_WRITE),
ioctl(LKST_IOC_MASKSET_SET), ioctl(LKST_IOC_MASKSET_DELETE),
ioctl(LKST_IOC_TRC_STATUS)

4.7.1.25. ioctl(LKST_IOC_MASKSET_DELETE)

<FUNCTION>

Delete a maskset

<SYNOPSIS>

```
#include <linux/lkst.h>
#include <linux/lkst_maskset.h>

int ioctl(int fd, int request, lkst_maskset_id id)
```

<ARGUMENTS>

fd	file descriptor(Return value opening LKST device.)
request	value 'LKST_IOC_MASKSET_ DELETE"
id	maskset ID

<RETURN VALUE>

0	success
EINVAL	A Special maskset is specified.
and/or	Specified maskset ID does not exist.
EBUSY	Specified maskset ID is currently selected.
EPERM	Was called by someone other than the superuser.

<DESCRIPTION>

Delete a specified maskset.
On success, this IOCTL deletes the maskset specified by the argument **id** and returns 0. Users cannot, however, delete a currently selected maskset. And masksets with IDs from 0 to 2 are special maskset and thus cannot be deleted.
On error (e.g., the specified maskset does not exist or a user tries to delete special maskset), this IOCTL returns a nonzero value described above.

<REFERENCES>

ioctl(LKST_IOC_MASKSET_READ), ioctl(LKST_IOC_MASKSET_WRITE),
ioctl(LKST_IOC_MASKSET_SET), ioctl(LKST_IOC_MASKSET_LIST)

4.7.1.26. ioctl(LKST_IOC_EVHANDLER_LIST)

<FUNCTION>

Return a list of registered event-handlers

<SYNOPSIS>

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

int ioctl(int fd, int request, struct lkst_evhandler_listparam *evhandler_listparam)
```

<ARGUMENTS>

fd	file descriptor(Return value opening LKST device.)
request	value 'LKST_IOC_ EVHANDLER_LIST"
evhandler_listparam	address of an **lkst_evhandler_listparam** structure object

```
struct lkst_evhandler_listparam {
        size_t      listent_size;                          /* size of the listent */
        struct lkst_evhandler_listent      *listent;       /* area to store the list of event handlers */
};

struct lkst_evhandler_listent {
         lkst_ evhandler_ id      id;                      /* event handler ID */
         char      name[LKST_EVHANDLER_NAME_LEN];          /* event handler name */
}
```

<RETURN VALUE>

0	success
ENOMEM	Kernel cannot allocate memory area to be used by this IOCTL.
EINVAL	Argument **listent** or **listent_size** is invalid.
EPERM	Was called by someone other than the superuser.

<DESCRIPTION>

Return the list of IDs and names of registered event-handlers.

The argument **evhandler_listparam** is the address of an **lkst_evhandler_listparam** structure object, the member **listent** specifies the area to which result is returned as a list of structure **lkst_evhandler_listent** type. In the list, this IOCTL stores entries in ascending order of event handler ID. The member **listent_size** specifies the size of the area (*). If **listent_size** is smaller than actual size of the list, this IOCTL stores the list up to the size of **listent_size**. Each event handler ID is stored as the member **id** of the **lkst_evhandler_listent** structure object, name of each event handler is stored as the member **name**.

On success, this IOCTL stores the list into **listent**, and returns 0.
On error, this IOCTL returns nonzero value described above. In this case, the values of the argument are not assured.
(*) To get **listent_size**, use **ioctl(LKST_IOC_TRC_STATUS)** for getting total number of event-handlers and then use **LKST_EVHANDLER_LISTENT_SIZE**([number of event-handlers]) macro.

<REFERENCES>
ioctl(LKST_IOC_EVHANDLER_CTRL), ioctl(LKST_IOC_TRC_STATUS)

4.7.1.27. ioctl(LKST_IOC_EVHANDLER_CTRL)

<FUNCTION>
Invoke an event-handler-control-function.

<SYNOPSIS>
```
#include <linux/lkst_evhandler.h>

int ioctl(int fd, int request, struct lkst_evhandler_ctrl_param *evhandler_ctrl_param)
```

<ARGUMENTS>

fd	file descriptor(Return value opening LKST device.)
request	value" LKST_IOC_EVHANDLER_CTRL"
evhandler_ctrl_param	address of an **evhandler_ctrl_param** structure object

```
struct lkst_evhandler_ctrl_param {
        lkst_ evhandler_id   id;          /* event handler ID */
        void      *buf;                   /* a communication area for control-function */
        size_t     bufsize;               /* size of the communication area */
        int      ret;                     /* return value from control-function */
}
```

<RETURN VALUE>

0	success
ENOMEM	Kernel cannot allocate memory area to be used by this IOCTL.
EINVAL	Specified event-handler ID is invalid.
and/or	Argument **buf** and/or **bufsize** is invalid.
and/or	Specified event-handler-control-function does not exist.
EPERM	Was called by someone other than the superuser.

<DESCRIPTION>
Invoke an event-handler-control-function of a specified event handler ID.
An event-handler-control-function to invoke is specified by the member **id** of an **lkst_evhandler_ctrl_param** structure object which the argument **evhandler_ctrl_param** points. The member **buf** and **bufsize** respectively specify the address of a communication area and the size of it. The communication area is used as an argument for the invoked function. Users store suitable values in the area according to the function.
On success, this IOCTL invokes the specified event-handler-control-function by taking the communication area as its argument. After the function is completed, this IOCTL stores the return value of the function as the member **ret** of the **lkst_evhandler_ctrl_param** structure object and returns 0.
On error (e.g., the specified event-handler-control-function does not exist), this IOCTL returns a nonzero value described above. In this case, the values of the argument are not assured.

<REFERENCES>
ioctl(LKST_IOC_EVHANDLER_LIST)

4.7.1.3 Buffer Control

4.7.1.31. ioctl(LKST_IOC_BUFFER_READ)

<FUNCTION>
Read a kernel-event buffer

<SYNOPSIS>
```
#include <linux/lkst.h>
#include <linux/lkst_buffer.h>

int ioctl(int fd, int request, struct lkst_log_buffer *lbuffer)
```

<ARGUMENTS>

fd	file descriptor(Return value opening LKST device.)
request	value "LKST_IOC_BUFFER_READ"
lbuffer	address of an **lkst_log_buffer** structure object

```
struct lkst_log_buffer {
        size_t read_size;                    /* the number of event records to read*/
        lkst_buffer_id id;                   /* processor number */
        size_t result_read_size;             /* the number of read event records */
        struct timeval xtime;                /* xtime */
        lkst_tsc_t tsc;                      /* machine cycle */
        lkst_cpu_freq_t cpu_freq;            /* cpu clockspeed in kHz */
        struct lkst_log_record *buffer;      /* address of a buffer to store event records */
        int endian_big;                      /* byte order, 0 if little endian */
        int buf_ver;                         /* LKST buffer version */
        char arch[LKST_ARCH_NAME_LEN]; /* Architecture name */
};

struct lkst_log_record {
        struct posix_log_entry posix;         /* log form specified by POSIX */
        lkst_arg_t log_arg1;                  /* 1st argument acquired at a trace point*/
        lkst_arg_t log_arg2;                  /* 2nd argument acquired at a trace point */
        lkst_arg_t log_arg3;                  /* 3rd argument acquired at a trace point */
        lkst_arg_t log_arg4;                  /* 4th argument acquired at a trace point */
}

struct posix_log_entry {
         unsigned int           log_magic;
         posix_log_recid_t      log_recid;       /* ID of the event record */
         size_t                 log_size;        /* size of the event record variable data */
         int                    log_format;      /* format of variable data */
         int                    log_event_type;  /* event identification code */
         posix_log_facility_t   log_facility;    /* event facility code */
         posix_log_severity_t   log_severity;    /* event severity code */
         uid_t                  log_uid;         /* effective user ID associated with the event */
         gid_t                  log_gid;         /* effective group ID associated with the event */
         pid_t                  log_pid;         /* process ID associated with event */
         pid_t                  log_pgrp;        /* process group associated with event */
         struct timespec        log_time;        /* event time stamp */
         unsigned int           log_flags;       /* bitmap of event flag */
         unsigned int           log_thread;      /* thread ID associated with event */
         posix_log_procid_t     log_processor    /* Processor ID associated with ev ent */
};
```

<RETURN VALUE>

0	success
EINVAL	Argument **buffer** and/or **read_size** is invalid.
and/or	Specified buffer ID is invalid.
and/or	Specified buffer does not exist.
EBUSY	Specified buffer is collapsed (or not initialized).
and/or	Reading mode of specified buffer is already set[2].
EPERM	Was called by someone other than the superuser.

<DESCRIPTION>

Events are recorded on the kernel-event buffers in order of the time when the events occurred. By using this IOCTL, users can get a list of the event log entries from the kernel-event buffers.

The size of event log entries (which size is **sizeof(struct lkst_log_record)**) to be read is specified by the member **read_size** (*) of an **lkst_log_buffer** structure object to which the argument **lbuffer** points. The member **buffer** specifies a virtual address of an area to which the event log entries are returned, and the member **id** specifies the ID of the kernel-event buffer from which the event log entries are read. However, if the specified buffer is in ring structure by ioctl(**LKST_IOC_BUFFER_RING**) or set reading mode by ioctl(**LKST_IOC_BUFFER_SETRMOD**), users should use read() systemcall to lkst device instead of using this IOCTL.

On success, this IOCTL reads the specified kernel-event buffer from the read pointer of the buffer, and stores a list of the event log entries in the area to which **buffer** points as a structure **lkst_log_record** type. When the specified kernel-event buffer is wrapped, this IOCTL reads from the oldest entry in the buffer. In case that reading process reaches end before reading up to **read_size**, this IOCTL stops reading and stores the actual size of read event log entries as the member **result_read_size** of the **lkst_log_buffer** structure object. Otherwise, **result_read_size** is equal to **read_size**. For each buffers, LKST stores a pair of struct timeval and machine cycle counter at the same time. the former represents time in the real world, and the latter can be compared with the time in the event buffers. Therefore, these and CPU frequency can be used as a compensation value for acquiring time stamp of the event log entry. In the members **xtime** , **tsc**, **cpu_freq** are copy of said timeval, machinecycle and CPU frequency. And in the members **endian_big** , **buf_ver**, **arch** are stored byte order, LKST buffer version and machine architecture name for analizing read events, if **endian_big** is 0, the byte order of the events are little endian. After them, this IOCTL updates the read pointer to a next event log entry after the one last read and returns 0.

On error, this IOCTL returns a nonzero value described above. In this case, the read pointer is not updated and the values of the argument are not assured.

(*)LKST internal kernel-event buffer is composed with entries which size is **LKST_SIZEOF_LKST_EVENT_RECORD** in byte. To get the value of argument **read_size**, please calculate the value by yourself. Following formula is an example;

```
        read_size = buffer_read_size * sizeof(struct lkst_log_record) /
                                                LKST_SIZEOF_LKST_EVENT_RECORD
```

[2] Currently this IOCTL would return it if this buffer is in ring structure

<Attention> If an event log entry has been overwritten while this IOCTL reads the record, the event type of the record is changed as **LKST_ETYPE_LKST_OVWRTN_REC**, and the original event type of the record is stored as 4th argument.

<REFERENCES>
ioctl(LKST_IOC_BUFFER_RING), ioctl(LKST_IOC_BUFFER_CREATE),
ioctl(LKST_IOC_BUFFER_DELETE), ioctl(LKST_IOC_BUFFER_SHIFT),
ioctl(LKST_IOC_BUFFER_LIST), ioctl(LKST_IOC_BUFFER_SETRMOD),

4.7.1.32. ioctl(LKST_IOC_BUFFER_CREATE)

<FUNCTION>
Create a new kernel-event buffer

<SYNOPSYS>
```
#include <linux/lkst.h>
#include <linux/lkst_buffer.h>

int ioctl(int fd, int request, struct lkst_buffer_param *buffer_param)
```

<ARGUMENTS>

fd	file descriptor (Return value opening LKST device.)
request	value "LKST_IOC_BUFFER_CREATE"
buffer_param	address of an **lkst_buffer_param** structure object

```
struct lkst_buffer_param {
        lkst_buffer_id id;          /* event buffer ID */
        int cpu;                    /* cpu number */
        size_t size;                /* size of kernel-event buffer */
        size_t result_size;         /* result size of kernel-event buffer */
};
```

<RETURN VALUE>

0	success
EINVAL	Specified buffer ID is invalid.
and/or	Specified buffer has already exist.
and/or	Specified CPU number is invalid.
and/or	**size** of the buffer is too small or large.
EBUSY	LKST has not been initialized (otherwise previous buffer of the specified buffer is collapsed by access violation).
and/or	No available Buffer ID.
and/or	Buffer list of specified CPU is already set reading mode.
ENOMEM	Kernel cannot allocate buffer area.
and/or	Memory area for the new buffer exceeds LKST available area.
EPERM	Was called by someone other than the superuser.

<DESCRIPTION>

This IOCTL creates a new kernel-event buffer in the kernel space. The ID of the buffer to be created is specified by the member **id** of an **lkst_buffer_param** structure object, that is pointed by an argument **buffer_param**. If **id** specifies **LKST_BUFFER_ID_VOID**, unused ID is allocated automatically. The allocated ID is sored in the member **id**. The CPU by which events are recorded into the new buffer is specified by the member **cpu**. However, if the buffer list of specified CPU is set reading mode by ioctl(**LKST_IOC_BUFFER_SETRMOD**), cannot add new buffer to the list. If the **cpu** specifies `-1`, new buffers are created for all CPUs. In this case, the **id** is ignored. The size of new buffer is specified by the member **size**. The value of **size** should be power of two (2), otherwize this IOCTL, however, does not return as error and assumes the size is the largest power-of-two number that does not exceed **size**.

On success, this IOCTL creates a new buffer and adds the buffer into the tail of a buffer list which correspond to the CPU specified by **cpu**. The allocated buffer size is stores as the member **result_size**. The new buffer, however, isn' t be selected as a buffer to record. This IOCTL, however, does not set the new buffer to ready to record. To start recording to the new buffer, use **ioctl(LKST_IOC_BUFFER_SHIFT)** with an ID of the buffer.

On error, this IOCTL returns nonzero value described above described above. In this case, this IOCTL doesn' t allocate memory for the new buffer.

<REFERENCES>

ioctl(LKST_IOC_BUFFER_RING), ioctl(LKST_IOC_BUFFER_READ),
ioctl(LKST_IOC_BUFFER_DELETE), ioctl(LKST_IOC_BUFFER_SHIFT),
ioctl(LKST_IOC_BUFFER_LIST), ioctl(LKST_IOC_BUFFER_SETRMOD),

4.7.1.33. ioctl(LKST_IOC_BUFFER_SHIFT)

<FUNCTION>

Switch currently selected kernel-event buffer to next one

<SYNOPSYS>

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

int ioctl(int fd, int request, int cpu)
```

<ARGUMENTS>

fd	file descriptor (Return value opening LKST device.)
request	value "LKST _IOC_BUFFER_SHIFT"
cpu	cpu number

<RETURN VALUE>

0	success
EINVAL	Specified CPU number is invalid.
EAGAIN	Buffers is now busy.
EINVAL	Next buffer of currently selected buffer does not exist.
EBUSY	LKST has not been initialized (otherwise currently selected buffer and/or next buffer of currently selected buffer is collapsed by access violation).
and/or	Buffer list of specified CPU is in ring structure.
and/or	Buffer list of specified CPU is already set reading mode.

<DESCRIPTION>

This IOCTL switch the buffer to record.

The CPU corresponding to the buffer to switch is specified by the member **cpu**. This IOCTL set a buffer pointed by the next pointer of the old current buffer as the new current buffer. If the buffer pointed by the next pointer does not exist, this IOCTL do nothing and returns as an error. If the buffer list of the specified CPU is in ring structure by ioctl(**LKST_IOC_BUFFER_RING**) or set reading mode by ioctl(**LKST_IOC_BUFFER_SETRMOD**), users cannot shift buffer of the list.

On success, this IOCTL switch the buffer and returns 0.

On error, this IOCTL returns nonzero value described above described above. In this case, this IOCTL does not switch the buffer.

<REFERENCES>

ioctl(LKST_IOC_BUFFER_RING), ioctl(LKST_IOC_BUFFER_READ),
ioctl(LKST_IOC_BUFFER_DELETE), ioctl(LKST_IOC_BUFFER_CREATE),
ioctl(LKST_IOC_BUFFER_LIST), ioctl(LKST_IOC_BUFFER_SETRMOD),

4.7.1.34. ioctl(LKST_IOC_BUFFER_DELETE)

<FUNCTION>

Delete a kernel-event buffer

<SYNOPSYS>

#include <linux/lkst.h>

#include <linux/lkst_buffer.h>

int ioctl(int fd, int request, lkst_buffer_id id)

<ARGUMENTS>

fd	file descriptor (Return value opening LKST device.)
request	value ‘LKST_IOC_BUFFER_DELETE"
id	kernel-event buffer id

<RETURN VALUE>

0	success
EINVAL	Cannot delete the buffer of ID=0.
and/or	Specified buffer ID is invalid.
EBUSY	Specified buffer ID is currently used.
and/or	Specified buffer is set reading mode.
EPERM	Was called by someone other than the superuser.

<DESCRIPTION>

Delete a specified kernel-event buffer.

On success, this IOCTL deletes the buffer specified by the argument **id**, maintains the link pointers in the list of kernel-event buffers and returns 0. Users cannot, however, delete a currently selected kernel-event buffer or the buffer which is set reading mode by ioctl(**LKST_IOC_BUFFER_SETRMOD**).And buffers with ID 0 are special buffer thus can not be deleted.

On error (e.g., the specified buffer does not exist or a user tries to delete special buffer), this IOCTL returns a nonzero value described above.

<REFERENCES>

ioctl(LKST_IOC_BUFFER_RING), ioctl(LKST_IOC_BUFFER_READ),
ioctl(LKST_IOC_BUFFER_SHIFT), ioctl(LKST_IOC_BUFFER_CREATE),
ioctl(LKST_IOC_BUFFER_LIST), ioctl(LKST_IOC_BUFFER_SETRMOD),

4.7.1.35. ioctl(LKST_IOC_BUFFER_LIST)

<FUNCTION>

Return a list of kernel-event buffers

<SYNOPSYS>

```
#include <linux/lkst.h>
#include <linux/lkst_buffer.h>

int ioctl(int fd, int request, struct lkst_buffer_listparam *buffer_listparam)
```

<ARGUMENTS>

fd	file descriptor (Return value opening LKST device.)
request	value ‘LKST_IOC_BUFFER_LIST"
buffer_listent	address of an **lkst_buffer_listparam** structure object

```
struct lkst_buffer_listparam {
        size_t listent_size;                    /* size of the listent */
        struct lkst_buffer_listent *listent;    /* area to store the list of event buffers */
};

struct lkst_buffer_listent {
        size_t size;                            /* buffer size */
        size_t readp;
        size_t writep;
        unsigned int wrap_flag;
        unsigned int current_flag;
        lkst_buffer_id id, prev, next;          /* buffer ID of own/prev/next buffer */
        int cpu;                                /* cpu number of the buffer */
};
```

<RETURN VALUE>

0	success
ENOMEM	Kernel cannot allocate memory area to be used by this ioctl.
EINVAL	Argument listent or listent_size is invalid.
and/or	Argument listent_size is too small or large.
EPERM	Was called by someone other than the superuser.

<DESCRIPTION>
Return a list of registered kernel-event buffers.
The argument **buffer_listparam** is the address of an **lkst_buffer_listparam** structure object, the member **listent** specifies the area to which result is returned as a list of structure **lkst_buffer_listent** type. In the list, this IOCTL stores entries in ascending order of buffer ID. The member **listent_size** specifies the size of the area (*). If **listent_size** is smaller than actual size of the list, this IOCTL stores the list up to the size of **listent_size**. Each buffer ID is stored as the member **id** of the **lkst_buffer_listent** structure object, and the buffer size is stored as the member **size**, a read pointer and a write pointer of the buffer are stored as the member **readp** and the member **writep**. LKST_FLAG_YES (=1) in the member **wrap_flag** indicates that the buffer has been wrapped around, and LKST_FLAG_YES in the member **current_flag** indicates that the buffer is used currently to record. LKST_FLAG_NO (=0) in each means opposit. The member **prev** and **next** represent a pair of link pointers in the buffer list. **LKST_BUFFER_ID_VOID** in **prev** and/or **next** indicates that linked buffer does not exists.
On success, this IOCTL stores the list into **listent**, and returns 0.
On error, This IOCTL returns a nonzero value described above. In this case, the values of the argument are not assured.
(*) To get **listent_size**, use **ioctl(LKST_IOC_TRC_STATUS)** for getting total number of kernel-event buffers and then use **LKST_BUFFER_LISTENT_SIZE(**[number of kernel-event buffers]**)** macro**.**

<REFERENCES>
ioctl(LKST_IOC_BUFFER_RING), ioctl(LKST_IOC_BUFFER_READ),
ioctl(LKST_IOC_BUFFER_SHIFT), ioctl(LKST_IOC_BUFFER_CREATE),
ioctl(LKST_IOC_BUFFER_DELETE), ioctl(LKST_IOC_BUFFER_SETRMOD),

4.7.1.36. ioctl(LKST_IOC_BUFFER_RING)

<FUNCTION>
Change structure of buffer list.

<SYNOPSYS>
#include <linux/lkst.h>
#include <linux/lkst_buffer.h>

int ioctl(int fd, int request, struct lkst_buffer_ringparam *rp)

<ARGUMENTS>

fd	file descriptor (Return value opening LKST device.)
request	value 'LKST_IOC_BUFFER_ RING"
rp	address of an **lkst_buffer_ringparam** structure object

```
struct lkst_buffer_ringparam {
        int cpu;            /* CPU number */
        int ring;           /* ring flag */
};
```

<RETURN VALUE>

0	success
EINVAL	Argument **<u>cpu</u>** or **<u>ring</u>** is invalid.
and/or	There is only one buffer in specified buffer list.
EBUSY	Specified buffer list is not initialized.
EPERM	Was called by someone other than the superuser.

<DESCRIPTION>

Change structure of buffer list to ring structure. This IOCTL is used by daemon process.

The argument **<u>rp</u>** is the address of an **lkst_buffer_ringparam** structure object, the member **<u>cpu</u>** specifies a CPU number correspond to a buffer list. If the member **<u>ring</u>** is 1, the structure is changed to ring list structure. If 0, the structure is changed to forward direction list. However, if there is only one buffer in the list, user cannot change the structure.

On success, this IOCTL changes structure of the buffer list, and returns 0.

On error, This IOCTL returns a nonzero value described above. In this case, structure of the buffer list is not changed.

<Attention> Reading or shifting operation for a buffer in the ring structured is forbidden.

<REFERENCES>

ioctl(LKST_IOC_BUFFER_LIST), ioctl(LKST_IOC_BUFFER_READ),
ioctl(LKST_IOC_BUFFER_SHIFT), ioctl(LKST_IOC_BUFFER_CREATE),
ioctl(LKST_IOC_BUFFER_DELETE), ioctl(LKST_IOC_BUFFER_SETRMOD),

4.7.1.37. ioctl(LKST_IOC_BUFFER_SETRMOD)

<FUNCTION>

Set reading mode of a buffer from opened virtual device.

<SYNOPSYS>

```
#include <linux/lkst.h>
#include <linux/lkst_buffer.h>

int ioctl(int fd, int request, struct lkst_buffer_srmodparam *sp)
```

<ARGUMENTS>

fd	file descriptor (Return value opening LKST device.)
request	value "LKST_IOC_BUFFER_ SETRMOD"
sp	address of an **lkst_buffer_srmodparam** structure object

```
struct lkst_buffer_srmodparam {
        int cpu;                                /* cpu number */
        int mode;                               /* reading mode */
        struct timeval xtime;                   /* xtime */
        lkst_tsc_t tsc;                         /* machine cycle */
        lkst_cpu_freq_t cpu_freq;               /* cpu clockspeed in kHz */
        int endian_big;                         /* byte order, 0 if little endian */
        int buf_ver;                            /* LKST buffer version */
        char arch[LKST_ARCH_NAME_LEN];          /* Architecture name */
};
```

<RETURN VALUE>

0	success
EINVAL	Argument **cpu** or **mode** is invalid.
EBUSY	Specified buffer list is already set by other process.
and/or	Caller process is already set to read other buffer list .
EPERM	Was called by someone other than the superuser.

<DESCRIPTION>

This IOCTL enables read() system call to opened virtual device by associating the virtual device and a specified buffer list and setting reading mode of a buffer in the buffer list. This IOCTL is used by daemon process.

The argument **sp** is the address of an **lkst_buffer_srmodparam** structure object, the member **cpu** specifies a CPU number correspond to a buffer list. The member **mode** specifies reading mode among **RAW/STD/SORTED**.(*)

On success, this IOCTL does above, and stores time stamp information, and return 0. In the members **xtime** , **tsc**, **cpu_freq** are stored timeval, machinecycle and CPU frequency for base time of recorded events. And in the members **endian_big** , **buf_ver**, **arch** are stored byte order, LKST buffer version and machine architecture name for analizing recorded events, if **endian_big** is 0, the byte order of the recorded events are little endian.(They were same as said in discripton of ioctl(**LKST_IOC_BUFFER_READ**).)
On error, This IOCTL returns a nonzero value described above. In this case, this IOCTL fails to set reading mode of buffer.
(*)Only **STD** mode is supported on ver 1.0.
<Attention> Reading, shifting, deleting, creating operation for a buffer which was set reading mode by this IOCTL is forbidden.

<REFERENCES>
ioctl(LKST_IOC_BUFFER_LIST), ioctl(LKST_IOC_BUFFER_READ),
ioctl(LKST_IOC_BUFFER_SHIFT), ioctl(LKST_IOC_BUFFER_CREATE),
ioctl(LKST_IOC_BUFFER_DELETE), ioctl(LKST_IOC_BUFFER_RING),

4.7.1.4 Trace Output

4.7.1.41. ioctl(LKST_IOC_ENTRY_LOG)
<FUNCTION>
Tell LKST that a specified event has occurred.

<SYNOPSIS>
```
#include <linux/lkst.h>

int ioctl(int fd, int request, struct lkst_entry_args *trace_arg)
```

<ARGUMENTS>

fd	file descriptor(Return value opening LKST device.)
request	value “LKST_IOC_ENTRY_LOG"
trace_arg	address of an **lkst_entry_args** structure object

```
struct lkst_entry_args {
        int event_type;                 /* corresponding event type */
        lkst_arg_t log_arg1;            /* 1st argument acquired at a trace point*/
        lkst_arg_t log_arg2;            /* 2nd argument acquired at a trace point */
        lkst_arg_t log_arg3;            /* 3rd argument acquired at a trace point */
        lkst_arg_t log_arg4;            /* 4th argument acquired at a trace point */
};
```

<RETURN VALUE>

0	success
EINVAL	Argument listent or listent_size is invalid.
EPERM	Was called by someone other than the superuser.

<DESCRIPTION>

Tell LKST that a specified event has occurred. This command always exits properly. Users can use this IOCTL as trace point.

An **lkst_entry_arg** structure object to which the argument **entry_arg** points specifies information for the entry, the member **event_type** specifies the type of event that is occurred, and the member **log_arg1**, **log arg2**, **log arg3**, and **log_arg4** specify 64bit-long values acquired at the trace point of the event

This IOCTL first checks whether the specified event handler ID is masked. If it is, this IOCTL returns 0 and exits.

If the specified event handler ID is not masked, this IOCTL invokes the event-handler-function corresponding to the specified even-handler ID. After finishing the function process, this IOCTL returns 0 and exits.

On error, This IOCTL returns a nonzero value described above. In ths case, this IOCTL fails to tell event occurring.

<REFERENCES>

IOCTL(LKST_IOC_MASKSET_LIST), IOCTL(LKST_IOC_EVHANDLER_LIST)

4.7.2 Kernel Functions

4.7.2.1 Congfoling LKST Status

4.7.2.11. lkst_trc_status()

<FUNCTION>

Return a current status of LKST.

<SYNOPSIS>

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

int lkst_trc_status(struct lkst_trc_status *trc_status)
```

<ARGUMENTS>

trc_status	address of an **lkst_trc_status** structure object

```
struct lkst_trc_status{
        lkst_maskset_id     current_maskset_id;        /* currently selected maskset ID */
        lkst_buffer_id      current_buffer_id[LKST_CPU_MAX];
                                                       /* currently selected buffer ID */
        int     maskset_num;                           /* total number of registered masksets */
        int     evhandler_num;                         /* total number of registered event handlers */
        int     buffer_num;                            /* total number of registered event buffers*/
};
```

<RETURN VALUE>

0	success
-EPERM	Was called by someone other than the superuser..

<DESCRIPTION>

This function provides the same function of IOCTL(LKST_IOC_TRC_STATUS)

<REFERENCES>

lkst_trc_start(), lkst_trc_stop(), lkst_maskset_list(), lkst_evhandler_list(), lkst_buffer_list(),

4.7.2.12. lkst_trc_statrt()

<FUNCTION>

Start LKST event tracing.

<SYNOPSIS>

#include <linux/lkst.h>

int lkst_trc_start()

<ARGUMENTS>

No arguments

<RETURN VALUE>

0	success
-EPERM	Was called by someone other than the superuser.

<DESCRIPTION>

This function provides the same function of IOCTL(LKST_IOC_TRC_START)

<REFERENCES>

lkst_trc_status(), ,lkst_trc_stop()

4.7.2.13. lkst_trc_stop()

<FUNCTION>

Stop LKST event tracing.

<SYNOPSIS>

#include <linux/lkst.h>

int lkst_trc_stop()

<ARGUMENTS>

No arguments

<RETURN VALUE>

0	success
-EPERM	Was called by someone other than the superuser.

<DESCRIPTION>
This function provides the same function of IOCTL(LKST_IOC_TRC_STOP)

<REFERENCES>
lkst_trc_status(), ,lkst_trc_start()

4.7.2.2 Maskset Contrl

4.7.2.21. lkst_maskset_read()

<FUNCTION>
Read contents of maskset.

<SYNOPSIS>
```
#include <linux/lkst.h>
#include <linux/lkst_maskset.h>
#include <linux/lkst_evhandler.h>

int lkst_maskset_read(struct lkst_maskset_param *maskset_param)
```

<ARGUMENTS>

maskset_param	address of an **lkst_maskset_param** structure object

```
struct lkst_maskset_param {
        lkst_maskset_id     id;                         /* maskset ID */
        size_t      maskset_size;                       /* maskset size*/
        struct lkst_maskset_body     *maskset           /* address of a maskset contents returned area */
};

struct lkst_maskset_body {
        char      name[LKST_MASKSET_NAME_LEN];     /* maskset name */
        lkst_maskset_table_len      len;           /* total number of maskset entries */
        struct lkst_maskset_entry   entry[LKST_MASKSET_TABLE_LEN_MAX];
                                                   /* maskset entry */
}

struct lkst_maskset_entry {
        int event_type;                            /* corresponding type of event */
        lkst_evhandler_id     id;                  /* event handler ID */
}
```

<RETURN VALUE>

0	success
-ENOMEM	Kernel cannot allocate memory area to be used by this function.
-EINVAL	Argument **<u>maskset</u>** or **<u>maskset_size</u>** is invalid.
and/or	Specified maskset ID (**<u>id</u>**) is invalid.
and/or	Specified maskset does not exist.
-EPERM	Was called by someone other than the superuser.

<DESCRIPTION>

This function provides the same function of ioctl(LKST_IOC_MASKSET_READ).

<REFERENCES>

lkst_maskset_write(), lkst_maskset_set(), lkst_maskset_list(), lkst_maskset_delete()

4.7.2.22. lkst_maskset_write()

<FUNCTION>

Register a new maskset

<SYNOPSIS>

```
#include <linux/lkst.h>
#include <linux/lkst_maskset.h>
#include <linux/lkst_evhandler.h>

int lkst_maskset_write(struct lkst_maskset_param *maskset_param)
```

<ARGUMENTS>

maskset_param	address of an **lkst_maskset_param** structure object

```
struct lkst_maskset_param {
        lkst_maskset_id     id;                      /* maskset ID */
        size_t    maskset_size;                      /* maskset size*/
        struct lkst_maskset_body    *maskset         /* address of a maskset stored area */
};

struct lkst_maskset_body {
        char      name[LKST_MASKSET_NAME_LEN];       /* maskset name */
        lkst_maskset_table_len     len;              /* total number of maskset entries*/
        struct lkst_maskset_entry   entry[LKST_MASKSET_TABLE_LEN_MAX];
                                                     /* maskset entry */
}
```

```
struct lkst_maskset_entry {
        int event_type;                         /* corresponding type of event */
        lkst_evhandler_id    id;                /* event handler ID */
}
```

<RETURN VALUE>

0	success
-ENOMEM	Kernel cannot allocate memory area to be used by this function.
and/or	Memory area for the new maskset exceeds LKST available area.
-EINVAL	Argument maskset or maskset_size is invalid.
and/or	Specified maskset ID is invalid.
and/or	Specified **event_type** is invalid.
and/or	Specified event-handler ID is invalid.
and/or	Specified event-handler does not exist.
and/or	Specify to record lock events with waking daemon process up .
-EBUSY	Specified maskset is collapsed (Overwrite case).
and/or	No available Maskset ID.
-EPERM	Was called by someone other than the superuser.

<DESCRIPTION>
This function provides the same function of ioctl(LKST_IOC_MASKSET_WRITE).

<REFERENCES>
lkst_maskset_read(), lkst_maskset_set(), lkst_maskset_list(), lkst_maskset_delete()

4.7.2.23. lkst_maskset_set()

<FUNCTION>
Switch a currently selected maskset

<SYNOPSIS>

```
#include <linux/lkst.h>
#include <linux/lkst_maskset.h>

int lkst_maskset_set(lkst_maskset_id id)
```

<ARGUMENTS>

id	maskset ID

<RETURN VALUE>

0	success
-EINVAL	Specified new maskset ID is invalid.
and/or	Specified new maskset does not exist.
-EBUSY	Currently selected maskset is not initialized.
and/or	Try to change maskset while LKST is stopped.
-EPERM	Was called by someone other than the superuser.

<DESCRIPTION>
This function provides the same function of ioctl(LKST_IOC_MASKSET_SET).

<REFERENCES>
lkst_maskset_read(), lkst_maskset_write(), lkst_maskset_list(), lkst_maskset_delete(),
lkst_trc_stop()

4.7.2.24. lkst_maskset_list()

<FUNCTION>
Return a list of registered masksets

<SYNOPSIS>
```
#include <linux/lkst.h>
#include <linux/lkst_maskset.h>

int lkst_maskset_list(struct lkst_maskset_listparam *maskset_listparam)
```

<ARGUMENTS>

maskset_listparam	address of an **lkst_maskset_listparam** structure object

```
struct lkst_maskset_listparam {
        lkst_maskset_id                 current_id;     /* current maskset ID */
        size_t      listent_size;                       /* size of the listent */
        struct lkst_maskset_listent     *listent        /* area to store the list of masksets */

};

struct lkst_maskset_listent {
        lkst_maskset_id     id;                         /* maskset ID */
        char      name[LKST_MASKSET_NAME_LEN];          /* maskset name */
        lkst_maskset_table_len      len;                /* total number of maskset entries */
}
```

<RETURN VALUE>

0	success
-ENOMEM	Kernel cannot allocate memory area to be used by this function.
-EINVAL	Argument **listent** and/or **listent_size** is invalid.
-EPERM	Was called by someone other than the superuser.

<DESCRIPTION>
This function provides the same function of ioctl(LKST_IOC_MASKSET_LIST).

<REFERENCES>
lkst_maskset_read(), lkst_maskset_write(), lkst_maskset_set(), lkst_maskset_delete(), lkst_trc_status()

4.7.2.25. lkst_maskset_delete()

<FUNCTION>
Delete a maskset

<SYNOPSIS>

```
#include <linux/lkst.h>
#include <linux/lkst_maskset.h>

int lkst_maskset_delete(lkst_maskset_id id)
```

<ARGUMENTS>

id	maskset ID

<RETURN VALUE>

0	success
-EINVAL	A Special maskset is specified.
and/or	Specified maskset ID does not exist.
-EBUSY	Specified maskset ID is currently selected.
-EPERM	Was called by someone other than the superuser.

<DESCRIPTION>
This function provides the same function of ioctl(LKST_IOC_MASKSET_DELETE).

<REFERENCES>
lkst_maskset_read(), lkst_maskset_write(), lkst_maskset_set(), lkst_maskset_list()

4.7.2.3 Event Handler Control

4.7.2.31. lkst_evhandler_list()

<FUNCTION>

Return a list of registered event handlers

<SYNOPSIS>

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

int lkst_evhandler_list(struct lkst_evhandler_listparam *evhandler_listparam)
```

<ARGUMENTS>

evhandler_listparam	address of an **lkst_evhandler_listparam** structure object

```
struct lkst_evhandler_listparam {
        size_t      listent_size;                              /* size of the listent */
        struct lkst_evhandler_listent      *listent;           /* area to store the list of event handlers */
};

struct lkst_evhandler_listent {
          lkst_ evhandler_ id      id;                         /* event handler ID */
          char      name[LKST_EVHANDLER_NAME_LEN];             /* event handler name */
}
```

<RETURN VALUE>

0	success
-ENOMEM	Kernel cannot allocate memory area to be used by this fuction.
-EINVAL	Argument **listent** or **listent_size** is invalid.
-EPERM	Was called by someone other than the superuser.

<DESCRIPTION>

This function provides the same function of ioctl(LKST_IOC_EVHANDLER_LIST).

<REFERENCES>

lkst_evhandler_ctrl(), lkst_evhandler_register(), lkst_trc_status()

4.7.2.32. lkst_evhandler_ctrl()

<FUNCTION>

Invoke an event-handler-control-function.

<SYNOPSIS>

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

int lkst_evhandler_ctrl(struct lkst_evhandler_ctrl_param *evhandler_ctrl_param)
```

<ARGUMENTS>

evhandler_ctrl_param	address of an **evhandler_ctrl_param** structure object

```
struct lkst_evhandler_ctrl_param {
        lkst_ evhandler_id     id;              /* event handler ID */
        void       *buf;                        /* a communication area for control-function */
        size_t      bufsize;                    /* size of the communication area */
        int       ret;                          /* return value from control-function */
}
```

<RETURN VALUE>

0	success
-ENOMEM	Kernel Cannot allocate memory area to be used by this function.
-EINVAL	Specified event-handler ID is invalid.
and/or	Argument **buf** and/or **bufsize** is invalid.
and/or	Specified event-handler-control-function does not exist.
-EPERM	Was called by someone other than the superuser.

<DESCRIPTION>

This function provides the same function of ioctl(LKST_IOC_EVHANDLER_CTRL).

<REFERENCES>

lkst_evhandler_list(), lkst_evhandler_register()

4.7.2.33. lkst_evhandler_register()

<FUNCTION>

Register an event-handler-function and an event-handler-control-function

<SYNOPSIS>

```
#include <linux/lkst_evhandler.h>
#include <linux/lkst_private.h>

int lkst_evhandler_register(lkst_ evhandler_id *id, char *name,
                            void *evhandler, int *evhandler_ctrl)
```

<ARGUMENTS>

id	event handler ID
name	event handler name
evhandler	address of an event-handler-function
evhandler_ctrl	address of an event-handler-control-function

<RETURN VALUE>

0	success
-EINVAL	Specified event-handler ID is invalid.
-EPERM	Was called by someone other than the superuser.

<DESCRIPTION>

Register an event-handler-function and an event-handler-control-function to a specified event handler ID.
The event-handler-function and the event-handler-control-function to register must be defined as prescribed format (*1).
The argument **id** specifies the event handler ID to register (*2). The argument **name** specifies the name of the event handler. The argument **evhandler** and **evhandler_ctrl** respectively specify the addresses of the event-handler-function and the event-handler-control-function. To register no function or to delete already registered function, these arguments must specify NULL value.
On success, this function registers the specified event-handler-function and event-handler-control-function to the specified event handler ID and returns 0.
On error, this function returns a nonzero value described above.
(Attention) The current event-handler-function or event-handler-control-function is overwritten.

(*)Prescribed format of event-handler-function and event-handler-control-function

<event-handler-function>

void [function name](void ***phookrec**, int **event_type**,
lkst_arg_t **log_arg1**, lkst_arg_t **log_arg2**, lkst_arg_t **log_arg3**, lkst_arg_t **log_arg4**)

Argument:

phookrec	reserved argument for GKHI (do not use in event handler function)
event_type	type of event;
log_arg1, log_arg2, log_arg3, log_arg4	arguments acquired at a trace point

<event-handler-control-function>

int [function name](void ***buf**, size_t **bufsize**)

arguments:

buf	address of a communication area
bufsize	size of the communication area

return value:

On success, return 0.

On error, returns a nonzero number.

(*2) The ID number of event handler is allocated as follows

0x000-0x02f	for the handlers that standard LKST provides.
0x030-0x07f	for user use
0x080-0x0ef	for the handlers provided by LKST addon modules
0x0f0-0x0ff	reserved for LKST internal use

<REFERENCES>

lkst_evhandler_list(), lkst_evhandler_ctrl()

4.7.2.4 Buffer Control

4.7.2.41. lkst_buffer_read()

<FUNCTION>

Read a kernel event buffer

<SYNOPSIS>

#include <linux/lkst.h>

#include <linux/lkst_buffer.h>

int lkst_buffer_read(struct lkst_log_buffer *lbuffer)

<ARGUMENTS>

lbuffer   address of an **lkst_log_buffer** structure object

```
struct lkst_log_buffer {
        size_t read_size;                       /* the number of event records to read*/
        lkst_buffer_id id;                      /* processor number */
        size_t result_read_size;                /* the number of read event records */
        struct timeval xtime;                   /* xtime */
        lkst_tsc_t tsc;                         /* machine cycle */
        lkst_cpu_freq_t cpu_freq;               /* cpu clockspeed in kHz */
        struct lkst_log_record *buffer;         /* address of a buffer to store event records */
        int endian_big;                         /* byte order, 0 if little endian */
        int buf_ver;                            /* LKST buffer version */
        char arch[LKST_ARCH_NAME_LEN]; /* Architecture name */
};

struct lkst_log_record {
        struct posix_log_entry posix;           /* log form specified by POSIX */
        lkst_arg_t log_arg1;                    /* 1st argument acquired at a trace point*/
        lkst_arg_t log_arg2;                    /* 2nd argument acquired at a trace point */
        lkst_arg_t log_arg3;                    /* 3rd argument acquired at a trace point */
        lkst_arg_t log_arg4;                    /* 4th argument acquired at a trace point */
}

struct posix_log_entry {
         unsigned int            log_magic;
         posix_log_recid_t       log_recid;       /* ID of the event record */
         size_t                  log_size;        /* size of the event record variable data */
         int                     log_format;      /* format of variable data */
         int                     log_event_type;  /* event identification code */
         posix_log_facility_t    log_facility;    /* event facility code */
         posix_log_severity_t    log_severity;    /* event severity code */
         uid_t                   log_uid;         /* effective user ID associated with the event */
         gid_t                   log_gid;         /* effective group ID associated with the event  */
         pid_t                   log_pid;         /* process ID associated with event */
         pid_t                   log_pgrp;        /* process group associated with event */
         struct timespec         log_time;        /* event time stamp */
         unsigned int            log_flags;       /* bitmap of event flag */
         unsigned int            log_thread;      /* thread ID associated with event */
         posix_log_procid_t      log_processor    /* Processor ID associated with event */
};
```

<RETURN VALUE>

0	success
-EINVAL	Argument **<u>buffer</u>** and/or **<u>read_size</u>** is invalid.
and/or	Specified buffer ID is invalid.
and/or	Specified buffer does not exist.
-EBUSY	Specified buffer is collapsed(or not initialized).
and/or	Specfied buffer is in ring structure.
and/or	Reading mode of specified buffer is already set.
-EPERM	Was called by someone other than the superuser.

<DESCRIPTION>

This function provides the same function of ioctl(LKST_IOC_BUFFER_READ).

<REFERENCES>

lkst_buffer_ring(), lkst_buffer_create(), lkst_buffer_delete(), lkst_buffer_shift), lkst_buffer_list(),lkst_buffer_setrmod()

4.7.2.42. lkst_buffer_create()

<FUNCTION>

Create a new kernel-event buffer

<SYNOPSYS>

```
#include <linux/lkst.h>
#include <linux/lkst_buffer.h>

int lkst_buffer_create(struct lkst_buffer_param *buffer_param)
```

<ARGUMENTS>

buffer_param address of an **lkst_buffer_param** structure object

```
struct lkst_buffer_param {
        lkst_buffer_id id;          /* event buffer ID */
        int cpu;                    /* cpu number */
        size_t size;                /* size of kernel-event buffer */
        size_t result_size;         /* result size of kernel-event buffer */
};
```

<RETURN VALUE>

0	success
-EINVAL	Specified buffer ID is invalid.
and/or	Specified buffer has already exist.
and/or	Specified CPU number is invalid.
and/or	**size** of the buffer is too small or large.
-EBUSY	LKST has not been initialized (otherwise previous buffer of the specified buffer is collapsed by access violation).
and/or	No available Buffer ID.
and/or	Buffer list of specified CPU is already set reading mode.
-ENOMEM	Kernel cannot allocate buffer area.
and/or	Memory area for the new buffer exceeds LKST available area.
-EPERM	Was called by someone other than the superuser.

<DESCRIPTION>

This function provides the same function of ioctl(LKST_IOC_BUFFER_CREATE).

<REFERENCES>

lkst_buffer_read(), lkst_buffer_ring(), lkst_buffer_delete(), lkst_buffer_shift(), lkst_buffer_list(),lkst_buffer_setrmod()

4.7.2.43. lkst_buffer_shift()

<FUNCTION>

Switch currently selected kernel-event buffer to next one

<SYNOPSYS>

#include <linux/lkst_buffer.h>

int lkst_buffer_shift(int cpu)

<ARGUMENTS>

cpu	cpu number

<RETURN VALUE>

0	success
-EINVAL	Specified CPU number is invalid.
-EAGAIN	Buffers is now busy.
-EINVAL	Next buffer of currently selected buffer does not exist.
-EBUSY	LKST has not been initialized (otherwise currently selected buffer and/or next buffer of currently selected buffer is collapsed by access violation).
and/or	Buffer list of specified CPU is in ring structure.
and/or	Buffer list of specified CPU is already set reading mode.

<DESCRIPTION>

This function provides the same function of ioctl(LKST_IOC_BUFFER_SHIFT).

<REFERENCES>

lkst_buffer_read(), lkst_buffer_ring(), lkst_buffer_delete(), lkst_buffer_create(), lkst_buffer_list(),lkst_buffer_setrmod()

4.7.2.44. lkst_buffer_delete()

<FUNCTION>
Delete a kernel-event buffer

<SYNOPSYS>
#include <linux/lkst.h>
#include <linux/lkst_buffer.h>

int lkst_buffer_delete(lkst_buffer_id id)

<ARGUMENTS>

id	kernel-event buffer id

<RETURN VALUE>

0	success
-EINVAL	Cannot delete the buffer of ID=0.
and/or	Specified buffer ID is invalid.
-EBUSY	Specified buffer ID is currently used.
and/or	Specified buffer is set reading mode.
-EPERM	Was called by someone other than the superuser.

<DESCRIPTION>
This function provides the same function of ioctl(LKST_IOC_BUFFER_DELETE).

<REFERENCES>
lkst_buffer_read(), lkst_buffer_ring(), lkst_buffer_create(), lkst_buffer_shift(), lkst_buffer_list(),lkst_buffer_setrmod()

4.7.2.45. lkst_buffer_list()

<FUNCTION>
Return a list of kernel-event buffers

<SYNOPSYS>
#include <linux/lkst.h>
#include <linux/lkst_buffer.h>

int lkst_buffer_list(struct lkst_buffer_listparam *buffer_listparam)

<ARGUMENTS>

buffer_listent	address of an **lkst_buffer_listparam** structure object

```
struct lkst_buffer_listparam {
        size_t listent_size;                    /* size of the listent */
        struct lkst_buffer_listent *listent;    /* area to store the list of event buffers */
};

struct lkst_buffer_listent {
        size_t size;                            /* buffer size */
        size_t readp;
        size_t writep;
        unsigned int wrap_flag;
        unsigned int current_flag;
        lkst_buffer_id id, prev, next;          /* buffer ID of own/prev/next buffer */
        int cpu;                                /* cpu number of the buffer */
};
```

<RETURN VALUE>

0	success
-ENOMEM	Kernel cannot allocate memory area to be used by this ioctl.
-EINVAL	Argument listent or listent_size is invalid.
and/or	Argument listent_size is too small or large.
-EPERM	Was called by someone other than the superuser.

<DESCRIPTION>

This function provides the same function of ioctl(LKST_IOC_BUFFER_LIST).

<REFERENCES>

lkst_buffer_read(), lkst_buffer_ring(), lkst_buffer_delete(), lkst_buffer_shift(), lkst_buffer_create(),lkst_buffer_setrmod()

4.7.2.46. lkst_buffer_ring()

<FUNCTION>

Change structure of buffer list.

<SYNOPSYS>

```
#include <linux/lkst.h>
#include <linux/lkst_buffer.h>

int lkst_buffer_ring(struct lkst_buffer_ringparam *rp)
```

<ARGUMENTS>

rp	address of an **lkst_buffer_ringparam** structure object

```
struct lkst_buffer_ringparam {
        int cpu;            /* CPU number */
        int ring;           /* ring flag */
};
```

<RETURN VALUE>

0	success
-EINVAL	Argument **cpu** or **ring** is invalid.
and/or	There is only one buffer in specified buffer list.
-EBUSY	Specified buffer list is not initialized.
-EPERM	Was called by someone other than the superuser.

<DESCRIPTION>

This function provides the same function of ioctl(LKST_IOC_BUFFER_RING).

<REFERENCES>

lkst_buffer_read(), lkst_buffer_list(), lkst_buffer_delete(), lkst_buffer_shift(), lkst_buffer_create(),lkst_buffer_setrmod()

4.7.2.47. lkst_buffer_setrmod()

<FUNCTION>

Set reading mode of buffer from opened virtual device.

<SYNOPSYS>

```
#include <linux/lkst.h>
#include <linux/lkst_buffer.h>

int lkst_buffer_setrmod(struct lkst_buffer_srmodparam *sp)
```

<ARGUMENTS>

sp	address of an **lkst_buffer_srmodparam** structure object

```
struct lkst_buffer_srmodparam {
        int cpu;                                /* cpu number */
        int mode;                               /* reading mode */
        struct timeval xtime;                   /* xtime */
        lkst_tsc_t tsc;                         /* machine cycle */
        lkst_cpu_freq_t cpu_freq;               /* cpu clockspeed in kHz */
        int endian_big;                         /* byte order, 0 if little endian */
        int buf_ver;                            /* LKST buffer version */
        char arch[LKST_ARCH_NAME_LEN];          /* Architecture name */
};
```

<RETURN VALUE>

0	success
-EINVAL	Argument **cpu** or **mode** is invalid.
-EBUSY	Specified buffer list is already set by other process.
and/or	Caller process is already set to read other buffer list .
-EPERM	Was called by someone other than the superuser.

<DESCRIPTION>

This function provides the same function of ioctl(LKST_IOC_BUFFER_SETRMOD).

<REFERENCES>

lkst_buffer_read(), lkst_buffer_list(), lkst_buffer_delete(), lkst_buffer_shift(), lkst_buffer_create(),lkst_buffer_ring()

4.7.3 User Commands

4.7.3.1 Controling LKST status

4.7.3.11. lkst

<NAME>

lkst - Control status of LKST.

<SYNOPSIS>

lkst command

<DESCRIPTION>

This command controls status of LKST. This start or stop event tracing, and display a current status such as number of masksets, buffers, event-handlers, id of currently selected maskset, and buffer of all CPUs.

<COMMANDS>

all	Outputs a current status and lists of all buffers, masksets, and event-handlers.
stat/status	Output a current status.
start	Start event tracing.
stop	Stop event tracing.
version/ver	Print version information.
help	Print help message.

<RETURN VALUE>

0	success
Except 0	failure

<REFERENCES>

lkstm, lkstbuf, lksteh,
ioctl(LKST_IOC_TRC_STATUS), ioctl(LKST_IOC_BUFFER_LIST),
ioctl(LKST_IOC_MASKSET_LIST), ioctl(LKST_IOC_EVHANDLER_LIST)

4.7.3.2 Maskset Control

4.7.3.21. lkstm

<NAME>

lkstm - Control maskset in LKST.

<SYNOPSIS>

lkstm command [option(s)]

<DESCRIPTION>

This command controls maskset in LKST, such as reading, writing, deletion, changing, and displaying list of all masksets.

<COMMANDS>

all Output a list of masksets and display content of all masksets.

exchange/xchg -m maskset_id

Change currently selected maskset. And it displays an id of old maskset.

<option>

-m maskset_id

Specify an id of a maskset to be selected.

(Attention) This command is not implemented yet.

delete/del -m maskset_id

Delete a maskset.

<option>

-m maskset_id

Specify an id of a maskset to be deleted.

list/ls

Output a list of all masksets.

read [-m maskset_id] [-A] [-a] [-d]

Output a content of maskset.

<options>

-m maskset_id

Specify an id of a maskset to be read. If omitted, a currently selected maskset is read.

-A

Read all masksets.

-a

Do not omit not registered event type. It ignored if "-d" option is specified.

-d

Do not display a description of each event type.

set -m maskset_id

Same as ' exchange' except for not displaying an id old maskset.

write [-m maskset_id] [-f file_name] [-n maskset_name] [-S]

Write a new maskset.

<options>

-m maskset_id

Specify an id of the maskset to be written. If omitted, empty id is selected automatically.

-f file_name

Specify a file which the content of the maskset is written. If ommited, standard input is used as input. This file can be created by "read" command as template.

-n maskset_name

Specify a name of the maskset. If omitted, the name of the maskset is set as "new_maskset".

-S

Change a currently selected maskset to the written maskset.

version/ver

Print version information.

help Print help message.

<RETURN VALUE>

0	success
Except 0	failure

<REFERENCES>

lkst,
ioctl(LKST_IOC_MASKSET_READ), ioctl(LKST_IOC_MASKSET_WRITE)
ioctl(LKST_IOC_MASKSET_LIST), ioctl(LKST_IOC_MASKSET_SET),
ioctl(LKST_IOC_MASKSET_DELETE)

4.7.3.3 Buffer Control

4.7.3.31. lkstlogd

<NAME>

lkstlogd - Linxu Kernel State Traccer logging utility.

<SYNOPSIS>

lkstlogd [-a] [-b buffer_size] [-l limit_size] [-n number] [-hv]

<DESCRIPTION>

lkstlogd supports to analyze faults which do not crash kernel; typically faults as follows:

A certain application can never start.
A certain daemon process ends suddenly, but the cause is unknown.

lkstlogd provides functions as follows:

When a specific signal is sent to lkstlogd, lkstlogd starts to write event logs, which is recorded by lkst, to a specified file. And lkstlogd continues writing the file until the signal is sent again.

In addition, when another special signal is sent, lkstlogd saves currently writing file and creates new one.

<OPTIONS>

-a

Start to write event logs to a file with starting lkstlogd.

-b buffer_size

Specify buffer size to create at initialization. (Default 2MByte)

-l limit_size

Specify a maximum size of log file in byte.
If the file size has reached the maximum size, lkstlogd rewinds the write pointer to head of the file.
The default size if 10MByte.

-n number

Specify number of buffer (for each CPU) to create at initialization. (Default 2)

-f log_file_name

Specify log file name.

-h

Print help message.

-v

Print version information.

<OUTPUT FORMAT>

Same as output of lkstbuf command.

<SIGNALS>

lkstlogd receives signals. To send signal to lkstlogd, use `kill` command as follow.

```
kill -SIGNAL `cat /var/run/lkstlogd.pid`
```

SIGHUP

Re-initialize lkstlogd. All the opened files are closed, and all child processes are terminated. Then lkstlogd is restarted.

SIGTERM

Terminate lkstlogd.

SIGUSR1

Start to write event logs to a file.
When lkstlogd receives it again, lkstlogd stops writing logs.

SIGUSR2

Change a file to write event logs.

<FILES>

/var/log/lkst/sebuf<cpu_number>.<serial_number>

lkstlogd writes event logs to different files for each CPU.
In addition, when lkstlogd receives SIGUSR2, saves currently writing file, and create new file of which serial_number is increased by 1, and start writing to the file.
This file can be changed by using '-f' option.

/var/run/lkstlogd.pid

This file holds process id of lkstlogd.
lkstlogd checks presence of this file at first. If it is, lkstlogd exits as error.

/etc/sysconfig/lkstlogd

Configuration file for lkstlogd.

This file format is shown as follows:

```
# comment
LKSTLOGDOPTION="-l 8388608"
```

# at line head shows this line is comment sentence.

In LKSTLOGDOPTION, start options of lkstlogd are written.

This file is read and interpreted by rc file at system initialization.

<REFERENCES>

lkstbuf,

ioctl(LKST_IOC_BUFFER_READ)

4.7.3.32. lkstbuf

<NAME>

lkstbuf - Control kernel event buffer in LKST.

<SYNOPSIS>

lkstbuf command [option(s)]

<DESCRIPTION>

This command controls kernel event buffer in LKST such as creation, delettion, changing, and displaying a list of all buffers.

<COMMANDS>

shift [-c cpu_id]

Change currently selected buffer to next buffer.

<option>

-c cpu_id

Specify an id of a CPU of which buffer is changed. If omitted, buffer of all CPUs is changed.

The id must be same as described in "/proc/cpuinfo".

create [-b buffer_id] [-c cpu_id] -s size

Create a new kernel event buffer.

<options>

-b buffer_id

Specify an id of a buffer to be created. If omitted, the empty id is selected automatically.

-c cpu_id

Specify an id of a CPU of which the buffer is created. The id must be same as described in "/proc/cpuinfo". If omitted, new buffers are created for all CPUs. In this case, specified buffer_id is ignored.

-s size

Specify a size of the buffer in byte.

delete/del -b buffer_id

Delete a kernel event buffer.

<option>

-b buffer_id

Specify an id of a buffer to be deleted.

list/ls

Output a list of all kernel event buffers.

read [-b buffer_id] [-f output_file] [-n number]

Read a content of kernel event buffer(s).

<options>

-b buffer_id

Specify an id of a buffer to be read. If ommited, all buffer is read.

-f output_file

Specify an name of output file. If omitted, this command displays the content.

-n number

Specify the number of read entry of the buffer. If ommited, all entries of the buffer are read.

print [-b buffer_id | -f input_file] [-c cpu_id] [-e event_name ...] [-h] [-n entry_num] [-r]

Display LKST trace data.

<options>

-b buffer_id

Specify an id of a buffer to be read. If omitted, all buffer is read.

-f input_file

Specify the file name of binary trace data, which is created by using lkstbuf read or lkstlogd.

(Also use kerenl crash dump file by using LKCD.)

If omitted, kernel buffer is read directly.

-c cpu_id

Specify the cpu number.

-e event_name ...

Specify event filter. Enumerate the names of events which you want to trace.

(Event name list is displayed by option -h.) Each event is separated by comma.

example(1) Display only "system_call_entry" and "system_call_exit".

-e system_call_entry,system_call_exit

If you don' t want to trace some events, specify"!" followed by the names of the events, like "!system_call_entry". You can specify "all" to specify all events.

example(2) Display all events except for "system_call_entry" and "system_call_exit".

-e all,!system_call_entry,!system_call_exit

NOTE: Enumerated event name is evaluated from the left to the right. If both display and non-display are specified for the same event, the specification of the right side is given to priority.

-e all,!spin_lock => display all events except spin_lock

-e !spin_lock,all => display all events

-e !spin_lock,spin_lock => the same as "-e spin_lock"

-e spin_lock,!spin_lock => the same as "-e !spin_lock"

-h

Display command help. (Also event name list is displayed.)

-n entry_num

Specify the number of output entry.

-r

Reverse the order of output record. (default : time descending sort)

version/ver

Print version information.

help Print help message.

<RETURN VALUE>

0	success
Except 0	failure

<REFERENCES>

lkst,
ioctl(LKST_IOC_BUFFER_READ), ioctl(LKST_IOC_BUFFER_LIST),
ioctl(LKST_IOC_BUFFER_SHIFT), ioctl(LKST_IOC_BUFFER_CREATE),
ioctl(LKST_IOC_BUFFER_DELETE)

4.7.3.4 Event-handler Control

4.7.3.41. lksteh

<NAME>

lksteh - Control event-handler in LKST.

<SYNOPSIS>

lksteh command [option(s)]

<DESCRIPTION>

This command controls event-handler in LKST, such as displaying a list of event-handler and invoking an event-handler-control-function.

<COMMANDS>

control/ctrl/c -e event_handler_id [-f file_name]

Invoke an event-handler-control-function.

<options>

-e event_handler_id

Specify an id of the event-handler-control-function to be invoked.

-f file_name

Specify an name of input file in which the data to be passed to the function is written.

If omitted, NULL data is passed.

list/ls

Output a list of all event-handlers.

version/ver

Print version information.

help Print help message.

<RETURN VALUE>

0	success
Except 0	failure

<REFERENCES>

lkststat,

ioctl(LKST_IOC_EVHANDLER_CTRL), ioctl(LKST_IOC_EVHANDLER_LIST)

4.7.3.5 Trace Output

4.7.3.51. lkstlogger

<NAME>

lkstlogger - Tells LKST that a specified event has occurred.

<SYNOPSIS>

lkstlogger command

lkstlogger -ev event_type [-a1 data1] [-a2 data2] [-a3 data3] [-a4 data4]

<DESCRIPTION>

This command tells LKST that a specified event has occurred. This command always exits properly. Users can use this command as trace point.

<COMMANDS>

version/ver

    Print version information.

help

    Print help message.

<OPTIONS>

-ev event_type

    Specify an occurred event type.

-a1 data1 -a2 data2 -a3 data3 -a4 data4

    Specify variable data to be transferred to LKST. If ommited, the data is specified as zero.

<RETURN VALUE>

0	Success
Except 0	failure

<REFERENCES>

ioctl(LKST_IOC_ENTRY_LOG)

4.7.3.6 Log formatter

4.7.3.61. logformat.pl

<NAME>

logformat.pl - Display LKST trace data.

<SYNOPSIS>

logformat.pl [-c cpu_id] [-e event_name ...] [-h] [-n entry_num] [-r] filename

<DESCRIPTION>

logformat.pl is a Perl Script that displays LKST trace data.
Argument "filename" must be specified the file name of binary trace data, which is created by using lkstbuf read or lkstlogd. (Also use kerenl crash dump file by using LKCD.)

<OPTIONS>

-c cpu_id

Specify the cpu number.

-e event_name ...

Specify event filter.
Enumerate the names of events which you want to trace.
(Event name list is displayed by option -h.) Each event is separated by comma.

example(1) Display only "system_call_entry" and "system_call_exit".

```
-e system_call_entry,system_call_exit
```

If you don' t want to trace some events, specify "!" followed by thenames of the events, like "!system_call_entry". You can specify "all" to specify all events.

example(2) Display all events except for "system_call_entry" and "system_call_exit".

```
-e all,!system_call_entry,!system_call_exit
```

NOTE:
Enumerated event name is evaluated from the left to the right.
If both display and non-display are specified for the same event, the specification of the right side is given to priority.

```
-e all,!spin_lock => display all events except spin_lock
-e !spin_lock,all => display all events
-e !spin_lock,spin_lock => the same as "-e spin_lock"
-e spin_lock,!spin_lock => the same as "-e !spin_lock"
```

-h

Display command help.

(Also event name list is displayed.)

-n entry_num

Specify the number of output entry.

-r

Reverse the order of output record.

(default : time descending sort)

<REFERENCES>

lkstbuf, lkstlogd.