Linux Kernel State Tracing Facility

Function Specifications

Version 02-03

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).
1.06	2002.08.30	hatasaki	- Modify description about how to insert trace points(in 4.6) - Modify description about lkst_evhandler_register() (in 4.7.2) - Add descriptions about some event handler functions (in 4.7.2). - Modify description about lkstbuf command (in 4.7.3)
1.07	2002.12.05	hiramatsu	- Modify descriptions about structure of the event buffer(in 4.2.2) - Add and Modify descriptions about IOCTLs(in 4.7.1)
1.08	2003.07.24	hiramatsu	- Add and Modify descriptions about LKST_ARG(in 4.6) - Modify descriptions about lkstbuf(in 4.7.3.32)
2.00	2003.10.27	hiramatsu	- Modify LKST block diagram (in 3) - Modify descriptions about LKST_ETYPE_DEF. (in 4.6) - Add descriptions about lkst_eh_device_register()/unregister(). (in 4.2.7.3) - Add descriptions about lkst_hook_etype_register()/unregister(). (in 4.2.7.5)

2.01	2003.12.24	Hiramatsu, sugita	- Modify descriptions about new event definition (in 4.6) - Add descriptions about –E option for lkstbuf and lkstm(in 4.7.3.2, 4.7.3.3) - Fix a description of buffer size. (in 4.2.2) - Correct the errors of the spelling.
2.02	2004.07.04	hiramatsu	- Add descriptions about procfs entry(in 4.3.2)
2.03	2004.08.24	hiramatsu	- Fix return values of the Kernel Functions - Separate the errno values and the return value of IOCTLs

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 INTERFACES 6

4.3.1 Device interface 6

4.3.2 Procfs interface 7

4.4 LOG OUTPUT AND FORMATTING 8

4.5 INITIALIZATION 8

4.6 INSERT NEW TRACE POINTS 9

4.7 FUNCTIONS 13

4.7.1 Device Interface 13

4.7.2 Kernel Functions 48

4.7.3 User Commands 74

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, modifications of important variables, and hardware events. And in addition to recording these events, it records information related to them.

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

  The logs recorded by LKST 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 save space of storages.

  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.

- Support 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

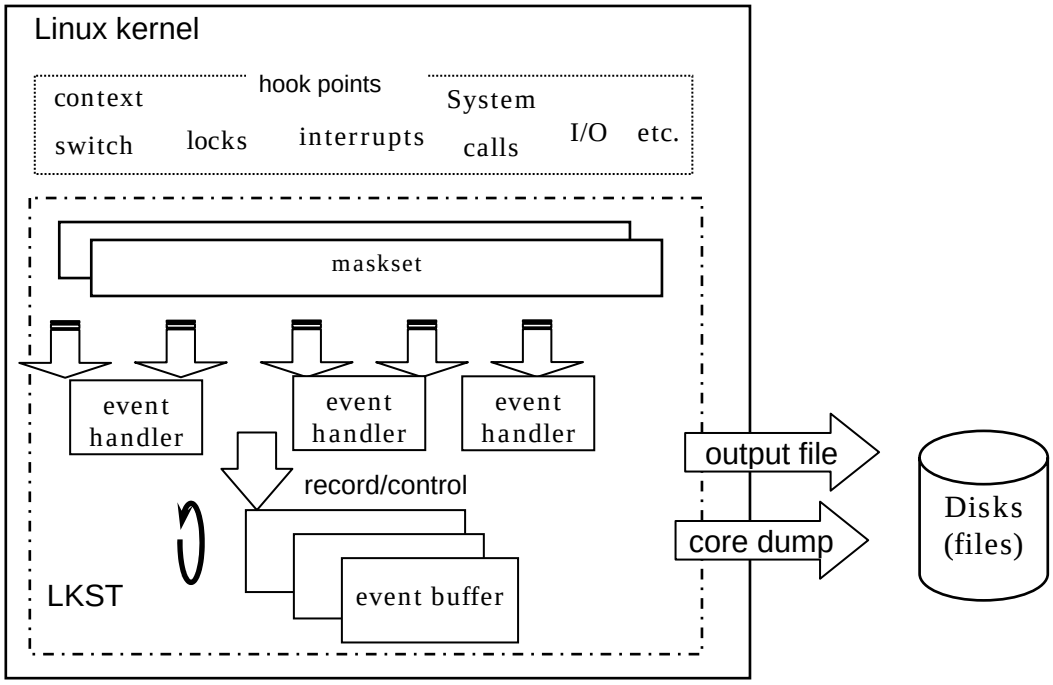

Figure 3-1 LKST block diagram.

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.
- 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 changed 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.

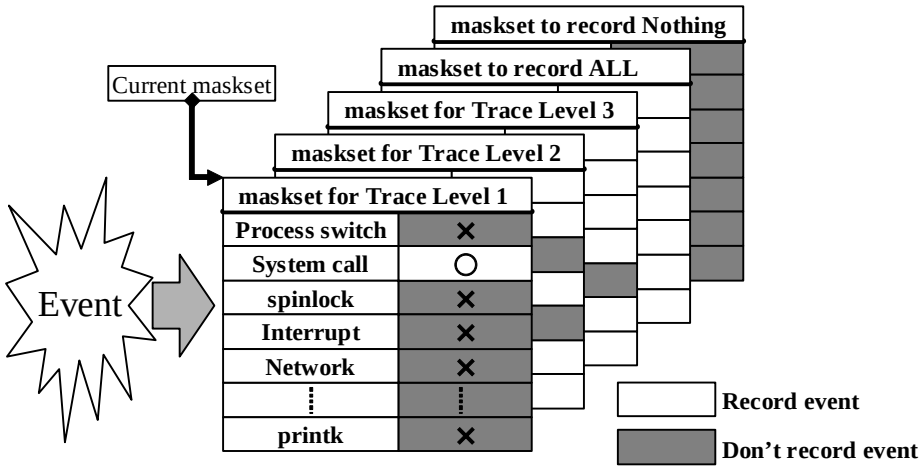

Figure 4-1 Masksets

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 a function that compresses events in the event buffer.

4.2.2 Structure of the event buffer

The event buffer is actually several sets of circular buffers. To avoid lock overhead in multiprocessor systems, each CPU has a different table of buffers. 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.

Each buffer has an ID indicating the ID of next buffer. Said IOCTL and the kernel function shifts current buffer to the buffer indicated by the next ID. Furthermore, you can change the buffer to record directly by giving an ID.

There are two limitations of the buffer. The first is that the size of a buffer will be limited as times of page size (architecture-depended, in IA32, this is 4KB). 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.

LKST has a buffer which allocated statically for special purpose. LKST uses it to record events occurred in initialization procedure, or events of LKST internal error. This buffer is not assigned any ID and written by a special API and a special handler nor able to delete.

4.3 Interfaces

4.3.1 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 process that issued ioctl(LKST_IOC_BUFFER_SETRMOD) can issue read().

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

4.3.2 Procfs interface

LKST has a proc file interface to provide information of events. You can read the information of events from /proc/lkst_etypes as below:

```
0x001,PROCESS_CONTEXTSWITCH,0x0001,"context_switch","pointer to task_struct prev ",
"pointer to task_struct next ","process state","process count"
0x002,PROCESS_WAKEUP,0x0001,"process_wakeup","pointer to wakeup-process p ",
"process state","synchronus",""
0x003,PROCESS_SIGSEND,0x0001,"process_sigsend","signal number sig ",
"pointer to process to which signal is sent","pointer to info structure info ",""
...
```

The format of this file is CSV (Comma Separated Value). Each column means event-ID, event-mnemonic, event-type flag, event-name, the description of 1st argument, the description of 2nd argument, the description of 3rd argument, and the description of 4th argument respectively.
The "event-type flag" is the bitmap of flags. The least significant bit means that the event is "maskable". In other words, if an event is "maskable", you can choose whether to trace the event or not by using the maskset. The second bit means that the event is used in many places in kernel.

This "event-type flag" (i.e. the combination of two bitmap flags) corresponds to hook-type of the event as below:

hook-type	event-type flag	Meanings
UNDEF	0x0002	An UNDEF hook-type event may be logged at many places. And user can't be changed whether it record or not by maskset.
NORMAL	0x0001	A NORMAL hook-type event is logged at specified place. And user can be change the event-handler by maskset.
INLINE	0x0003	An INLINE hook-type event may be logged at many places. And user can be change the event-handler by maskset.
MODULE	0x0003	The MODULE hook-type is same as the INLINE.

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
	Kernel debugging	Performance evaluation, etc	OS failure	Service failure	
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 masksets 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

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 below:

(1) Define a new event: If you want to use a new event, you should define it before use. LKST supports several ways to define a new event. Here is a table (Table 4-4) that helps determining how and where to define it.

Table 4-4 Macros and Definitions for Event

The event is caused ...	Including the kernel binary			Only kernel module(s)		
	One core function (one location)	Functions capable to build as module(s)	Inline function, or several locations	One module (one location)	Several modules	Inline function, or several locations
Definition place	(Arch. depended) include/asm-*/lkst_etypes.h (Others/General routine) include/linux/lkst_etypes.h			The same module	The module loaded first. (the root module of module's dependency tree.)	
Definition macro	LKST_ETYPE_DEF			LKST_ETYPE_DEF_MODULE (*)		
Hook type (**)	NORMAL	MODULE	INLINE	NORMAL	MODULE	INLINE
Hook macro	LKST_HOOK	LKST_HOOK_INLINE		LKST_HOOK	LKST_HOOK_INLINE	

(*) When using this macro, you should initialize and terminate new event in proper functions. For more detail, see following 1.b) section.

(**) If the event that never used by any LKST_HOOK* macro (for example, the event is passed to lkst_evhandlerprim_entry_log() directory), define it as UNDEF hook-type.

For more detail, see following sections.

1.a) If new trace points into kernel binary (vmlinux) with a new event, define new event type in linux/lkst_events.h as follows:

```
LKST_ETYPE_DEF(event-ID, hook-type, mnemonic, event name string,
                    arg1 description, arg2 desc., arg3 desc., arg4 desc.)
```

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

hook-type --- Type of hook header for Kernel Hooks

Specify either the following according to the inserting
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
  (or capable to be 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.
- UNDEF: If you insert HOOK macro in the module
  (or capable to be in the module),
  use this type.

mnemonic --- Mnemonic of the event type

event name string, arg1 description..
--- event name, and argument data descriptions for log formatter.

(example)

```
LKST_ETYPE_DEF(0x100, NORMAL, NEW_EVENT,
                "user added event","data1", "data2", "data3", "data4")
```

1.b) If the trace points are ONLY in the modules, the macro for declaration of HOOK header can be put in the prime module (IOW, the root module of dependency tree). And the functions for the initialization and termination of HOOK are inserted respectively in the functions of the module-initialization and module-termination.

Declaration macro for HOOK

```
LKST_ETYPE_DEF_MODULE(event-ID, hook-type, event-symbol, \
                "event-name", \
                "1st argument description", \
                "2nd argument description", \
                "3rd argument description", \
                "4th argument description")
```

NOTE: you can specify NORMAL to hook-type if this event is ONLY caused at the same module and at one location.

Initialization function

lkst_hook_etype_register(&LKST_ETYPE_INFO(event-symbol));

Termination function

lkst_hook_etype_unregister(&LKST_ETYPE_INFO(event-symbol));

(Example)

```
LKST_ETYPE_DEF_MODULE(0x100, NORMAL, NEW_EVENT,
                                    “user added event”, “data1”, “data2”, ”data3”, ”data4”)
static int __init testmod_init(void)
{
    int ret;
    ret = lkst_hook_etype_register(&LKST_ETYPE_INFO(NEW_EVENT));

    return ret;
}
static void __exit testmod_exit(void)
{
    lkst_hook_etype_unregister(&LKST_ETYPE_INFO(NEW_EVENT));
}
module_init(testmod_init);
module_exit(testmod_exit);
```

(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 acquired 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)))
  Ands If user use LKST_ARG() macro, it expand bit-width depends on
  CONFIG_DEBUG_LKST_DONOT_EXPAND_ARGBITS kernel configuration.
   LKST_ARG(data)

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

4.7.1 Device Interface

All IOCTL commands must be called by the superuser.

4.7.1.1 Controlling LKST Status

4.7.1.1.1. 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_status_param {
        unsigned long   online_cpu;                     /* bitmap of online cpus*/
        lkst_maskset_id     current_maskset_id;         /* current selected maskset ID */
        lkst_buffer_id      write_buf[LKST_CPU_MAX]; /* current writing buffer ID */
        lkst_buffer_id      read_buf[LKST_CPU_MAX];  /* current reading buffer ID */
        int                 maskset_num;            /* total number of registered masksets */
        int                 evhandler_num;          /* total number of registered event handlers */
        int         static_buffer_recid;            /* recid of the static buffer*/
        size_t      static_buffer_size;             /* size of the static buffer*/
};
```

<RETURN VALUE>

0	Success
non-zero	Failure

<ERROR>

EINVAL	Argument **trc_status** is invalid
and/or	Failed to execute copy_to/from_user().
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. The **online_cpu** has a bitmap describing which CPUs are online. A currently selected maskset ID is stored in the **current_maskset_id**. Currently writing kernel-event buffer IDs for each CPUs are stored in the corresponding entries of the **write_buf**. array. Also reading buffer IDs are stored in the **read_buf** array. The total number of registered masksets is stored in the **maskset_num**. The total number of event handlers is stored in the **evhandler_num**. The **static_buffer_recid** and the **static_buffer_size** denote the recid of the static shared buffer and the size respectively.
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)

4.7.1.1.2. 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
non-zero Failure

<ERROR>
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),

4.7.1.1.3. 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
non-zero Failure

<ERROR>
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

4.7.1.2.1. 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
non-zero	Failure

<ERROR>

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 specified 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)

4.7.1.2.2. 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
non-zero	Failure

<ERROR>

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 maskset name is not specified.
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 stored 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)

4.7.1.2.3. 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
non-zero	Failure

<ERROR>

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)

4.7.1.2.4. 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
non-zero	Failure

<ERROR>

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)

4.7.1.2.5. 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
non-zero	Failure

<ERROR>

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)

4.7.1.2.6. 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
non-zero	Failure

<ERROR>

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)

4.7.1.2.7. 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
non-zero	Failure

<ERROR>

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

4.7.1.3.1. 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 {
        int cpu;                                /* cpu number */
        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 clock speed 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
non-zero	Failure

<ERROR>

EINVAL	Argument **buffer** and/or **read_size** is invalid.
and/or	Specified buffer ID is invalid.
and/or	Specified buffer does not exist.
and/or	Failed to execute copy_to/from_user().
EPERM	Was called by someone other than the superuser.
ENOMEM	Kernel does not have enough memory to operate.

<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** and the member **cpu** specify the ID and the CPU of the kernel-event buffer from which the event log entries are read respectively. Especially, when **id** is LKST_BUFFER_ID_VOID and **cpu** is '-1', this IOCTL reads from static shared buffer.

On success, this IOCTL reads the specified kernel-event buffer from head 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. In case that reading process reaches end before reading up to **read_size** or is caught up by writing process, 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 analyzing read events, if **endian_big** is 0, the byte order of the events are little endian. After them, this IOCTL 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
```

<REFERENCES>

ioctl(LKST_IOC_BUFFER_LINK), ioctl(LKST_IOC_BUFFER_CREATE),
ioctl(LKST_IOC_BUFFER_DELETE), ioctl(LKST_IOC_BUFFER_SHIFT),
ioctl(LKST_IOC_BUFFER_JUMP),ioctl(LKST_IOC_BUFFER_LIST), ioctl(LKST_IOC_BUFFER_SETRMOD),

4.7.1.3.2. 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 */
        lkst_buffer_id next;        /* event buffer ID of next buffer */
        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
non-zero	Failure

<ERROR>

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).
ENOSPC	No available Buffer ID.
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 sorted in the member **id**. The CPU by which events are recorded into the new buffer is specified by the member **cpu**. If the **cpu** specifies `-1`, new buffers are created for all CPUs. The size of new buffer is specified by the member **size**. The value of **size** should be times of 4KB, otherwise, this IOCTL treat the value of **size** as the largest times of 4KB number that does not exceed **size**. The member **next** specifies the buffer ID to which new buffer is linked. Note, you can specify the buffer that is not exist yet to the **next**. Until it is referred, LKST does never check whether it exists.

On success, this IOCTL creates a new buffer and inserts the buffer into the table of a buffer which corresponds to the CPU specified by **cpu**. The allocated buffer size is stores as the member **result_size**. The new buffer, however, isn't selected as a buffer to record yet. To start recording to the new buffer, use **ioctl(LKST_IOC_BUFFER_JUMP)** with an ID of the buffer and a CPU owns that 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_LINK), ioctl(LKST_IOC_BUFFER_READ),
ioctl(LKST_IOC_BUFFER_DELETE), ioctl(LKST_IOC_BUFFER_SHIFT),
ioctl(LKST_IOC_BUFFER_JUMP),ioctl(LKST_IOC_BUFFER_LIST), ioctl(LKST_IOC_BUFFER_SETRMOD),

4.7.1.3.3. 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
non-zero	Failure

<ERROR>

EINVAL	Specified CPU number is invalid.
and/or	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).

<DESCRIPTION>

This IOCTL switch the buffer to record.

The CPU corresponding to the buffer to switch is specified by the **cpu**. This IOCTL set a buffer pointed by the member **next** of the old current buffer as the new current buffer. If the buffer pointed by the **next** does not exist, this IOCTL do nothing and returns as an error.

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_LINK), 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),

4.7.1.3.4. ioctl(LKST_IOC_BUFFER_JUMP)

<FUNCTION>

Switch currently selected kernel-event buffer to specified one

<SYNOPSYS>

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

int ioctl(int fd, int request, lkst_buffer_jumpparam jump_param)
```

<ARGUMENTS>

fd file descriptor (Return value opening LKST device.)

request value “LKST_IOC_BUFFER_JUMP"

jump_param address of an **lkst_buffer_jumpparam** structure object

```
struct lkst_buffer_jumpparam {
        int cpu;                    /* cpu number */
        lkst_buffer_id dest;        /* destination buffer id */
};
```

<RETURN VALUE>

0	Success
non-zero	Failure

<ERROR>

EINVAL	Specified CPU number is invalid.
and/or	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).

<DESCRIPTION>

This IOCTL switch the buffer to record. (Likely to ioctl(LKST_IOC_BUFFER_SHIFT))

The CPU corresponding to the buffer to switch is specified by the member **cpu** of the **jump_param** argument. This IOCTL set a buffer pointed by the member **dest** of the **jump_param** argument as the new current buffer. If the buffer pointed by the **dest** does not exist, this IOCTL do nothing and returns as an error.

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_LINK), 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),
ioctl(LKST_IOC_BUFFER_SHIFT)

4.7.1.3.5. ioctl(LKST_IOC_BUFFER_LINK)

<FUNCTION>

Assign destination buffer used when performing buffer shift, to another buffer.

<SYNOPSYS>

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

int ioctl(int fd, int request, lkst_buffer_linkparam link_param)
```

<ARGUMENTS>

fd	file descriptor (Return value opening LKST device.)
request	value “LKST_IOC_BUFFER_LINK"
link_param	address of an **lkst_buffer_linkparam** structure object

```
struct lkst_buffer_linkparam {
        int cpu;                        /* cpu number */
        lkst_buffer_id id, next;        /* source and destination buffer id */
};
```

<RETURN VALUE>

0	Success
non-zero	Failure

<ERROR>

EINVAL	Specified CPU number is invalid.
and/or	destination buffer is the same buffer (link the buffer itself).
EBUSY	LKST has not been initialized (otherwise currently selected buffer and/or next buffer of currently selected buffer is collapsed by access violation).

<DESCRIPTION>

This IOCTL assigns destination buffer used when performing buffer shift, to another buffer.

The pair of the member **id** and **cpu** of the structure **lkst_buffer_linkparam** type specify an event-buffer. And the member **next** specifies the destination buffer to be switched from the buffer specified by former two members. When the **next** specifies **LKST_BUFFER_ID_VOID**, this IOCTL assigns the buffer to terminal buffer (the buffer does not have any destination buffer, so "buffer shift" operation always fails on this buffer). However if the **next** specifies **id** itself, this IOCTL fails operation.

You can make buffers ring structure with this IOCTL (instead of ioctl(**LKST_IOC_BUFFER_RING**)). Also you can make tree structure, and combination of both.

On success, this IOCTL assigns the buffer specified by the member **next**, to the buffer specified by the member **id**

and **cpu** and returns 0.

On error (e.g., the **next** specifies **id** itself), this IOCTL returns a nonzero value described above.

<REFERENCES>

ioctl(LKST_IOC_BUFFER_JUMP), 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),
ioctl(LKST_IOC_BUFFER_SHIFT)

4.7.1.3.6. 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_delparam del_param)
```

<ARGUMENTS>

fd	file descriptor (Return value opening LKST device.)
request	value “LKST_IOC_BUFFER_DELETE"
del_param	address of an **lkst_buffer_delparam** structure object

```
struct lkst_buffer_delparam {
        int cpu;                    /* cpu number */
        lkst_buffer_id id;          /* buffer id to be deleted */
};
```

<RETURN VALUE>

0	Success
non-zero	Failure

<ERROR>

EINVAL	Cannot delete the buffer of ID=0.
and/or	Specified buffer ID is invalid.
EBUSY	Specified buffer ID is currently used.
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 member **id** and the member **cpu** of the **del_param** and returns 0.

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_LINK), ioctl(LKST_IOC_BUFFER_READ),
ioctl(LKST_IOC_BUFFER_SHIFT), ioctl(LKST_IOC_BUFFER_SHIFT),

ioctl(LKST_IOC_BUFFER_CREATE),ioctl(LKST_IOC_BUFFER_LIST),
ioctl(LKST_IOC_BUFFER_SETRMOD),

4.7.1.3.7. 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 {
        int cpu;                                /* cpu number of the buffer */
        int listent_num;                        /* num of the listent */
        int buffer_num;                         /* num of the buffers */
        lkst_buffer_id write_buf;               /* current writing buffer */
        lkst_buffer_id read_buf;                /* current reading buffer */
        lkst_buffer_id rq_head;                 /* head of read queue */
        lkst_buffer_id rq_tail;                 /* tail of read queue*/
        pid_t owner;                            /* owner pid */
        int read_pos;                           /* reading position */
        struct lkst_buffer_listent *listent;    /* area to store the list of event buffers*/
};

struct lkst_buffer_listent {
        size_t size;                            /* buffer size */
        int write_offset;                       /* offset to write */
        int baseid;                             /* base counter */
        lkst_buffer_id id, next;                /* buffer ID of own/next buffer */
        lkst_buffer_id rq_prev, rq_next;        /* buffer ID of read-queue previous/next buffer */
};
```

<RETURN VALUE>

0 Success

non-zero Failure

<ERROR>

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 **cpu** specifies CPU by which buffers are owned. 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 fills entries in ascending order of buffer ID. The member **listent_num** specifies the number of the entries. If **listent_num** is smaller than actual number of the list, this IOCTL stores the list up to the number of **listent_num**. How many buffers are owned by this CPU is stored in the member **buffer_num** by this IOCTL. Also the IDs of currently recording buffer and reading buffer are stored in the member **write_buf** and **read_bnf** respectively. And the member **rq_head** and **rq_tail** mean the head of buffer read queue and tail respectively. The member **owner** is PID of read process executing read system call, when the value of **owner** is equal to PID_MAX+1, there is no read processes on this CPU(see ioctl(LKST_IOC_BUFFER_SETRMOD)). The value of the member **read_pos** means where you can begin to read from the buffer specified by **read_buf.** 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**. The member **write_offset** and **baseid** are used by generating recid (serial record id) of event-log entries. The member **rq_prev** and **rq_next** represent a pair of link pointers in the buffer read queue. And the member **next** represents the destination buffer of "shift" operation.

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.

<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),

4.7.1.3.8. ioctl(LKST_IOC_BUFFER_RING)

<CAUTION>

THIS FUNCTION IS NO LONGER SUPPORTED. INSTEAD OF USING THIS IOCTL, YOU CAN USE IOCTL(LKST_IOC_BUFFER_LINK) TO CREATE RING STRUCTURE. SEE IOCTL(LKST_IOC_BUFFER_LINK) FOR MORE DETAIL.

4.7.1.3.9. 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
non-zero	Failure

<ERROR>

EINVAL	Argument **cpu** or **mode** is invalid.
EBUSY	Specified buffer list is already assigned 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 process to perform read() system call by binding the process to specified CPU and makes the process into the "owner". This IOCTL is mainly used by daemon process. But also anyone who want to perform read() system call can use this IOCTL.

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**.(*)
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 analyzing recorded events, if **endian_big** is 0, the byte order of the recorded events are little endian.(They were same as said in description 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 1.4.

<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_LINK),

4.7.1.4 Trace Output

4.7.1.4.1. 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

non-zero Failure

<ERROR>

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 this case, this IOCTL fails to tell event occurring.

<REFERENCES>
IOCTL(LKST_IOC_MASKSET_LIST), IOCTL(LKST_IOC_EVHANDLER_LIST)

4.7.1.5 System calls(LKST specified)

4.7.1.5.1. read() system call

<FUNCTION>

read lkst virtual device.

<SYNOPSIS>

#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count);

<ARGUMENTS>

fd	file descriptor(Return value opening LKST device.)
buf	the address of user-side buffer.
count	the size to read.

<RETURN VALUE>

0 or more	Success (how many bytes we read)
less than 0	Failure

<ERROR>

EINVAL	the address of buffer or the size of buffer is invalid.
EBUSY	this process is not bound to any CPU yet.
EPERM	Was called by someone other than the superuser

<DESCRIPTION>

The read() system call reads entries from the read queue of buffers owned by the CPU that has been bound to reading process. The read-queue is a doubly linked list of buffers. Each CPU owns a different read-queue of cause. The buffers in the read-queue are ordered by writing. When the read() system call is called, read process starts reading from the buffer that positioned at the head of this read-queue. At the same time, the buffer is removed from the read-queue. If the process reads out the buffer and there is still space in **buf**, it try to continue to read from next buffer that positioned at the head of the read-queue. If the read-queue is empty, the process will wait to fill the read-queue. However, before calling read() system call, the process sets O_NONBLOCK to file descriptor with fctrl() system call, the process does not wait and returns immediately.

<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_LINK),

4.7.2 Kernel Functions

4.7.2.1 Control LKST Status

4.7.2.1.1. lkst_trc_status()

<FUNCTION>

Return a current status of LKST.

<SYNOPSIS>

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

int lkst_trc_status(struct lkst_status_param *trc_status)
```

<ARGUMENTS>

trc_status address of an **lkst_status_param** structure object

```
struct lkst_status_param {
        unsigned long   online_cpu;                     /* bitmap of online cpus*/
        lkst_maskset_id     current_maskset_id;         /* current selected maskset ID */
        lkst_buffer_id      write_buf[LKST_CPU_MAX]; /* current writing buffer ID */
        lkst_buffer_id      read_buf[LKST_CPU_MAX];  /* current reading buffer ID */
        int                 maskset_num;            /* total number of registered masksets */
        int                 evhandler_num;          /* total number of registered event handlers */
        int         static_buffer_recid;            /* recid of the static buffer*/
        size_t      static_buffer_size;             /* size of the static buffer*/
};
```

<RETURN VALUE>

0	Success
-EINVAL	The **trc_status** parameter is NULL.

<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(),

4.7.2.1.2. lkst_trc_statrt()

<FUNCTION>

Start LKST event tracing.

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

int lkst_trc_start()

<ARGUMENTS>
No arguments

<RETURN VALUE>
0 Success
-EAGAIN Can not get the **lkst_maskset_lock** lock. Try again.

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

<REFERENCES>
lkst_trc_status(), ,lkst_trc_stop()

4.7.2.1.3. lkst_trc_stop()

<FUNCTION>
Stop LKST event tracing.

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

int lkst_trc_stop()

<ARGUMENTS>
No arguments

<RETURN VALUE>
0 Success
-EAGAIN Can not get the **lkst_maskset_lock** lock. Try again.

<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

4.7.2.2.1. 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 **maskset** or **maskset_size** is invalid.
and/or	Specified maskset ID (**id**) is invalid.
and/or	Specified maskset does not exist.
--EAGAIN	Can not get the **lkst_maskset_lock** lock. Try again.

<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()

4.7.2.2.2. 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.
-EAGAIN	Can not get the **lkst_maskset_lock** lock. Try again.

<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()

4.7.2.2.3. 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.

-EAGAIN Can not get the **lkst_maskset_lock** lock. Try again.

<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()

4.7.2.2.4. 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
-EINVAL Argument **listent** and/or **listent_size** is invalid.
-EAGAIN Can not get the **lkst_maskset_lock** lock. Try again.

<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()

4.7.2.2.5. 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.
-EAGAIN	Can not get the **lkst_maskset_lock** lock. Try again.

<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()

4.7.2.2.6. lkst_maskset_get_id()

<FUNCTION>

Find a maskset ID by name.

<SYNOPSIS>

#include <linux/lkst_maskset.h>

#include <linux/lkst_private.h>

int lkst_maskset_get_id(const char *name)

<ARGUMENTS>

name the name of a maskset

<RETURN VALUE>

maskset ID	Success
-EINVAL	Specified maskset name is invalid.
-EAGAIN	Can not get the **lkst_maskset_lock** lock. Try again.

<DESCRIPTION>

Find maskset ID by specified maskset name.

On success, this function returns the maskset ID which has specified name. If the event-handler does not exist, it returns **LKST_MASKSET_ID_VOID**.

On error, this function returns a negative value described above.

<REFERENCES>

lkst_maskset_read(), lkst_maskset_write(), lkst_maskset_set(), lkst_maskset_list()

4.7.2.3 Event Handler Control

4.7.2.3.1. 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

-EINVAL Argument **listent** or **listent_size** is invalid.

-EAGAIN Can not get the **lkst_evhandler_lock** lock. Try again.

<DESCRIPTION>

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

<REFERENCES>

lkst_evhandler_ctrl(), lkst_evhandler_register(), lkst_evhandler_unregister(), lkst_evhandler_get_id(), lkst_trc_status()

4.7.2.3.2. 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
-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.

<DESCRIPTION>

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

<REFERENCES>

lkst_evhandler_list(), lkst_evhandler_register(), lkst_evhandler_unregister(), lkst_evhandler_get_id()

4.7.2.3.3. 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, const 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>

Registered event-handler ID	Success
-EINVAL	Specified event-handler ID is invalid.
and/or	Specified event-handler name is invalid.
and/or	Specified event-handler function is NULL.
-EAGAIN	Can not get the **lkst_evhandler_lock** lock. Try again.
-ENOSPC	No available event-handler ID.

<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). If **id** specifies **LKST_EVHANDLER_ID_VOID**, unused ID is allocated automatically. The argument **name** specifies the name of the event handler, and the **name** must be unique. The argument **evhandler** and **evhandler_ctrl** respectively specify the addresses of the event-handler-function and the event-handler-control-function.

On success, this function registers the specified event-handler-function and event-handler-control-function to the specified event handler ID and returns registered event handler ID.

On error, this function returns a negative value described above.

(*1)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 Kernel Hooks (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

0x000-0x01f reserved for LKST internal use.

0x020-0x0fffor user use

<REFERENCES>

lkst_evhandler_list(), lkst_evhandler_ctrl(), lkst_evhandler_unregister(), lkst_evhandler_get_id()

4.7.2.3.4. lkst_evhandler_unregister()

<FUNCTION>

Unregister an event-handler-function and an event-handler-control-function

<SYNOPSIS>

#include <linux/lkst_evhandler.h>

#include <linux/lkst_private.h>

int lkst_evhandler_unregister(lkst_ evhandler_id id)

<ARGUMENTS>

id event handler ID

<RETURN VALUE>

0	Success
-EINVAL	Specified event-handler ID is invalid.
-EAGAIN	Can not get the **lkst_evhandler_lock** lock. Try again.

<DESCRIPTION>

Deregister specified ID of an event-handler-function and an event-handler-control-function.

On success, this function deregisters the specified ID of event-handler-function and event-handler-control-function and returns 0.

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

(Attention) Users cannot deregister LKST reserved event-handlers.

<REFERENCES>

lkst_evhandler_list(), lkst_evhandler_ctrl(), lkst_evhandler_register()

4.7.2.3.5. lkst_evhandler_get_id()

<FUNCTION>

Find an event-handler ID by name.

<SYNOPSIS>

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

int lkst_evhandler_get_id(const char *name)
```

<ARGUMENTS>

name	event handler name

<RETURN VALUE>

event-handler ID	Success
-EINVAL	Specified event-handler name is invalid.
-EAGAIN	Can not get the **lkst_evhandler_lock** lock. Try again.

<DESCRIPTION>

Find event-handler ID by specified event-handler name.

On success, this function returns the event-handler ID which has specified event-handler name. If the event-handler does not exist, it returns **LKST_EVHANDLER_ID_VOID**.

On error, this function returns a negative value described above.

<REFERENCES>
lkst_evhandler_list(), lkst_evhandler_ctrl(), lkst_evhandler_register()

4.7.2.3.6. lkst_eh_device_register()

<FUNCTION>
Register an event-handler-device object

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

int lkst_eh_device_register(struct lkst_ eh_device * ehdev)

```
struct lkst_eh_device {
        char name[LKST_EVHANDLER_NAME]; /* the name of event handler*/
        int id;                                 /* the id of event handler */
        lkst_evhandler_func evhandler;
        lkst_evhandler_ctrl_func evhandler_ctrl;
        ssize_t (*ctrl_store)(struct lkst_eh_device *, const char *, size_t);
        ssize_t (*ctrl_show)(struct lkst_eh_device *, char *);
        struct class_device class;
}
```

<ARGUMENTS>
ehdev the event-handler-device object

<RETURN VALUE>

Registered event-handler ID	Success
-EINVAL	Specified object is NULL.
and/or	Specified object is not initialized correctly.
-ENOSPC	No available event-handler ID.
-EAGAIN	Can not get the **lkst_evhandler_lock** lock. Try again.

<DESCRIPTION>
Register an event-handler device object to the kobject driver model.
The event-handler device object contains properties that be needed to register an event-handler-function. To initialize these properties, you can use following macro:

LKST_EH_DEV_DEF(name, evh, evh_ctl, cstore, cshow)

name the name of the event-handler (symbol, **NOT** string)

evh the event handler function

evh_ctl the event handler control function

cstore the “store”(IOW, “write”) function for sysfs' “ctrl” attribute file.

cshow the “show”(IOW, “read”) function for sysfs' “ctrl” attribute file.

By default, this macro set the event-handler ID to LKST_EVHANDLER_ID_VOID.

If you want to specify that, do as like as following: LKST_EH_DEV(name).id = 0x100;

This function calls lkst_evhandler_register() function, and if it fail, this returns error code from that directly. Also, this function is based on kobject/sysfs new driver model. After this function called, the new event handler device appears under */sys/class/lkst/* .

For example, When you register a new event-handler named **NEW_LOG_HANDLER**, there is a new directory like this:

/sys/class/lkst/NEW_LOG_HANDLER/

And this directory has 3 attributes(files), *id*, *name*, and *ctrl*. The event-handler ID and the name are contained in the two former files, respectively. The *ctrl* file is a new interface from/to userspace. You can call **cstore**() function to write something to that file, and **cshow**() function to read from.

On success, this function registers the specified event-handler-function and event-handler-control-function to the specified event handler ID and returns registered event handler ID.

On error, this function returns a negative value described above.

<REFERENCES>

lkst_evhandler_list(), lkst_evhandler_ctrl(), lkst_evhandler_register(), lkst_evhandler_unregister(), lkst_evhandler_get_id()

4.7.2.3.7. lkst_eh_device_unregister()

<FUNCTION>

Unregister an event-handler-device object

<SYNOPSIS>

#include <linux/lkst_class.h>

void lkst_eh_device_unregister(struct lkst_ eh_device * ehdev)

<ARGUMENTS>

ehdev the event-handler-device object

<RETURN VALUE>

None

<DESCRIPTION>

Deregister specified event-handler-device object (and call lkst_evhandler_unregister() automatically).

(Attention) Users cannot deregister LKST reserved event-handler-device object.

<REFERENCES>

lkst_evhandler_list(), lkst_evhandler_ctrl(), lkst_evhandler_register(), lkst_evhandler_unregister(), lkst_evhandler_get_id()

4.7.2.4 Buffer Control

4.7.2.4.1. 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 clock speed 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 or more	Success (read size in bytes)
-EINVAL	Argument **buffer** and/or **read_size** is invalid.
and/or	Specified buffer ID is invalid.
and/or	Specified buffer does not exist.

<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()

4.7.2.4.2. lkst_buffer_create()

<FUNCTION>

Create a new kernel-event buffer

<SYNOPSYS>

#include <linux/lkst.h>

#include <linux/lkst_buffer.h>

int lkst_buffer_create(lkst_buffer_id id, size_t size, lkst_buffer_id next)

<ARGUMENTS>

lkst_buffer_id id	event buffer ID
size_t size;	the size of kernel-event buffer
lkst_buffer_id next	the id of buffer of destination.

<RETURN VALUE>

Buffer ID	Success
-EINVAL	Specified buffer has already exist.-
-ENOMEM	Kernel cannot allocate buffer area.
and/or	**size** of the buffer is too large.
--ENOSPC	There is no available buffer ID.

<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()

4.7.2.4.3. lkst_buffer_jump()

<FUNCTION>

Switch currently selected kernel-event buffer to specified one

<SYNOPSYS>

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

int lkst_buffer_jump(lkst_buffer_id next)
```

<ARGUMENTS>

cpu	cpu number

<RETURN VALUE>

0	Success
-EINVAL	Specified buffer id is invalid.
-EBUSY	LKST has not been initialized (otherwise currently selected buffer and/or next buffer of currently selected buffer is collapsed by access violation).

<DESCRIPTION>

This function provides the same function of ioctl(LKST_IOC_BUFFER_JUMP). However this can switch to only the buffer owned by the same CPU.

<REFERENCES>

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

4.7.2.4.4. lkst_buffer_shift()

<FUNCTION>

Switch currently selected kernel-event buffer to next one

<SYNOPSYS>

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

int lkst_buffer_shift(void)
```

<RETURN VALUE>

0	Success

-EINVAL The buffer referred by the member next of current buffer is invalid.
-EBUSY LKST has not been initialized (otherwise currently
selected buffer and/or next buffer of currently
selected buffer is collapsed by access violation).

<DESCRIPTION>
This function provides the same function of ioctl(LKST_IOC_BUFFER_SHIFT). However this can switch to only the buffer owned by the same CPU.

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

4.7.2.4.5. 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.

<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()

4.7.2.4.6. lkst_buffer_list()

<DEFUNCT>

THIS FUNCTION IS NO LONGER SUPPORTED. INSTEAD OF USING THIS FUNCTION, YOU CAN REFER ENTRIES OF THE lkst_cpu ARRAY DIRECTORY.

4.7.2.4.7. lkst_buffer_ring()

<DEFUNCT>

THIS FUNCTION IS NO LONGER SUPPORTED. INSTEAD OF USING THIS FUNCTION, YOU CAN MODIFY ENTRIES OF THE buffer_table ARRAY THAT IS THE MENBER OF THE lkst_current DIRECTORY.

<CAUTION>

!!YOU MUST NOT MODIFY ANY MEMBERS OF ANOTHER ENTRY OF lkst_cpu EXCEPT THE ENTRY DEFINED BY THE lkst_current MACRO!!

4.7.2.4.8. lkst_buffer_setrmod()

<DEFUNCT>

THIS FUNCTION IS NO LONGER SUPPORTED. NOW, SETRMOD FUNCTION IS JUST SUPPORTED BY IOCTL.

4.7.2.5 Event Types

4.7.2.5.1. lkst_hook_etype_register()

<FUNCTION>

Register an event-type information and enable it.

<SYNOPSIS>

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

int lkst_hook_etype_register(struct lkst_ etype_info * einfo)

struct lkst_etype_info {
        int etype;                      /* event-type */
        struct hook * hook;             /* the pointer of hook structure */
        struct hook_rec hook_rec;       /* hook_rec structure */
        char * mnemonic;                /* mnemonic string */
        char * name;                            /* name string */
        char * args[4];                 /* arguments string */
}
```

<ARGUMENTS>

einfo	the event-type information

<RETURN VALUE>

Registered event-handler ID	Success
-EINVAL	Specified einfo is NULL.
and/or	Specified einfo is not initialized correctly.
and/or	Specified einfo->id is reserved.
-EBUSY	Specified einfo->id is already used.

<DESCRIPTION>

Register an event-type information and enable it.

The event-type information contains properties that be needed to enable event. To define an **lkst_etype_info** data structure and to initialize those properties, you can use following macro:

```
LKST_ETYPE_INFO_DEF(etype, mnemonic, name, arg1, arg2, arg3, arg4, hook)
```

etype	the number of event-type (event-ID, integer)
mnemonic	the symbol of the event-type (symbol, **NOT** string)
name	the name of the event-type (string)
arg1...4	the descriptions of arguments (string)

hook	the pointer of the **hook** data structure.

This macro defines ONLY an **lkst_etype_info** data structure and doesn't define a **hook** data structure. So, instead of using that, using following macro is recommended.

LKST_ETYPE_DEF_MODULE(etype, mnemonic, name, arg1, arg2, arg3, arg4)

etype	the number of event-type (event-ID, integer)
mnemonic	the symbol of the event-type (symbol, **NOT** string)
name	the name of the event-type (string)
arg1...4	the descriptions of arguments (string)

This macro defines a symbol and 2 variables; an enum symbol that **LKST_ETYPE_mnemonic**, a **hook** data structure named **LKST_ETYPE_mnemonic_HEADER**, and an **lkst_etype_info** data structure named **lkst_einfo_mnemonic**. And this initializes those variables correctly.

After this function called, the new event-type information appears in */proc/lkst_etypes*.
On success, this function registers the specified event-type information to the specified event ID and returns registered event ID.
On error, this function returns a negative value described above.

<REFERENCES>
lkst_hook_etype_unregister()

4.7.2.5.2. lkst_hook_etype_unregister()

<FUNCTION>

Unregister an event-type information and disable it.

<SYNOPSIS>

#include <linux/lkst_hook.h>

int lkst_hook_etype_unregister(struct lkst_etype_info * einfo)

<ARGUMENTS>

einfo the event-type information

<RETURN VALUE>

-EINVAL	Specified einfo is NULL.
-EBUSY	Specified einfo->id is reserved id.
and/or	Specified einfo is not registered.

<DESCRIPTION>

Unregister specified event-type information and disable it.

(Attention) Users cannot unregister LKST reserved event-type information.

<REFERENCES>

lkst_hook_etype_register()

4.7.3 User Commands

4.7.3.1 Controling LKST status

4.7.3.1.1. 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

non-zero 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

4.7.3.2.1. 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.

delete/del -m maskset_id | -n maskset_name

Delete a maskset.

<options>

-m maskset_id

Specify the id of a maskset to delete.

-n maskset_name

Specify the id of a maskset to delete.

list/ls

Output a list of all masksets.

read [-m maskset_id | -n maskset_name] [-A] [-a] [-d] [-E etype_list_file]

Output a content of maskset.

<options>

-m maskset_id

Specify an id of a maskset to read.

-n maskset_name

Specify the id of a maskset to read. If both maskset_id adn maskset__name are omitted,

read currently selected maskset.

-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.

-E etype_list_file

Specify event-type listed file. If omitted, only embedded events are shown. You can get this list from /proc/lkst_etypes.

set -m maskset_id | -n maskset_name

Change currently selected maskset.

<options>

-m maskset_id

Specify the id of a maskset to select.

-n maskset_name

Specify the name of a maskset to select.

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 omitted, standard input is used as input. This file can be created by "read" command as template.

-n maskset_name

Specify the name of new maskset. If omitted, the name of new maskset is set as "new_maskset0". When it is already used, change to "new_maskset1". And when it is used too, change to "new_maskset2", and so on (in other words, this increments tag number).

-S

Change a currently selected maskset to the written maskset.

find -n maskset_name

Find the id of a maskset from its name.

<option>

-n maskset_name

Specify the name of a maskset to find.

NOTE:

If no maskset has maskset_name, then outputs 255(=LKST_MASKSET_ID_VOID).

config/conf [-m maskset_id | -n maskset_name] <event_type> <event-handler_id>

Config a maskset

<options>

-m maskset_id

Specify the id of a maskset to configure.

-n maskset_name

Specify the name of a maskset to configure. If both of id and name are omitted, currently selected

maskset is configured.

event_type

Specify the type of an event to change.

event-handler_id

Specify the id of an event-handler to change to.

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

4.7.3.3.1. 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[K | M]

Specify buffer size to create at initialization. buffer_size is followed by K and M suffixes representing size in Kilo bytes and Mega bytes, respectively. (Default 2MByte)

-l limit_size[K | M]

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.
limit_size is followed by K and M suffixes representing size in Kilo bytes and Mega bytes, respectively. 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)

4.7.3.3.2. 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, deletion, changing, reading, formatting, and listing of all buffers.

<COMMANDS>

link/ln -b buffer_id [-n next_buffer_id] [-c cpu_id]

Change destination buffer for the shift operation.

<options>

-b buffer_id

Specify the id of a buffer.

-n next_buffer_id

Specify the id of the destination buffer. If omitted, clear destination of specified buffer.

NOTE: You can specify a buffer doesn't exist yet to destination, except itself. However, you must fail to shift while it doesn't exist.

-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.

shift [-c cpu_id]

Change currently selected buffer to next buffer.

NOTE: If destination buffer does not exist, this operation would fail.

<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".

jump -b buffer_id [-c cpu_id]

Change currently selected buffer to specified buffer.

<options>

-b buffer_id

Specify the id of a buffer of destination.

-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] [-n next_buffer_id] -s size

Create a new kernel event buffer.
<options>
-s size[K|M]
Specify a size of the buffer. size is followed by K and M suffixes representing size in Kilo bytes and in Mega bytes, respectively.
-b buffer_id
Specify an id of a buffer to be created. If omitted, the empty id is selected automatically.
-n next_buffer_id
Specify the id of the destination buffer. If omitted, clear the destination of new buffer.
-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.

delete/del -b buffer_id [-c cpu_id]

Delete a kernel event buffer.
<options>
-b buffer_id
Specify an id of a buffer to be deleted.
-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.

list/ls [-c cpu_id] [-v]

Output a list of all kernel event buffers.
<options>
-c cpu_id
Specify the id of a CPU to list. The id must be same as described in "/proc/cpuinfo".
If omitted, all buffers those have buffer_id id are deleted in each CPU (if exist).
-v
Specify that the list of buffers display as verbose format.

read [-c cpu_id] [-b buffer_id] [-f output_file] [-n number] [-E etype_list_file]
read -S [-f output_file] [-n number] [-E etype_list_file]

Read the contents of an event buffer/event buffers.
<options>

-c cpu_id

Specify the id of a cpu to read from. If omitted, all buffers those have buffer_id id are read.

-b buffer_id

Specify the id of a buffer to be read. If omitted, all buffers owned by cpu_id are read.

Neither cpu_id nor buffer_id is specified, all buffers in system are read (except for static buffer)

-S

Specify that the command read from static buffer.

-f output_file

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

-n number

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

-E etype_list_file

Specify event-type listed file. If omitted, only embedded events are shown. You can get this list from /proc/lkst_etypes.

print -f input_file [-c cpu_id] [-e event_name ...] [-h] [-n entry_num] [-r] [-C [-S]] [-V] [-E etype_list_file]

Display LKST trace data.

<options>

-f input_file

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

(Also use kernel 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)

-E etype_list_file

Specify event-type listed file. If omitted, only embedded events are shown. You can get this list from /proc/lkst_etypes.

-V

Verbose mode. Display all arguments. (Default: display only commented arguments)

-C [-S]

Display trace data by CSV format.

The meaning of column in the CSV format is explained.

First column is event name or event number.

Second column is CPU number.

Third column is process id.

4th-8th columns are date which an event caused.

...,day of week, month, day, hour,year,...

9th column is name of parameter.

10th column is value of parameter.(under 32bit)

11th column is value of parameter.(upper 32bit)

... following ,same mean 9th-11th column.

when you specify -S option, print short format of date as follows(4th-5th columns)

...,second,usecond

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

4.7.3.4.1. 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

4.7.3.5.1. 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 omitted, 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

4.7.3.6.1. 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 kernel 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 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)

<REFERENCES>

lkstbuf, lkstlogd.