

SystemTap Tapset Reference Manual

SystemTap

SystemTap Tapset Reference Manual

by SystemTap

Copyright © 2008-2009 Red Hat, Inc. and others

This documentation is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

For more details see the file COPYING in the source distribution of Linux.

Table of Contents

1. Introduction	1
Tapset Name Format	1
2. Context Functions	2
function::print_regs	3
function::execname	4
function::pid	5
function::tid	6
function::ppid	7
function::pgroup	8
function::sid	9
function::pexecname	10
function::gid	11
function::egid	12
function::uid	13
function::euid	14
function::is_myproc	15
function::cpu	16
function::pp	17
function::registers_valid	18
function::user_mode	19
function::is_return	20
function::target	21
function::module_name	22
function::stp_pid	23
function::stack_size	24
function::stack_used	25
function::stack_unused	26
function::uaddr	27
function::cmdline_args	28
function::cmdline_arg	29
function::cmdline_str	30
function::env_var	31
function::print_stack	32
function::sprint_stack	33
function::probfunc	34
function::probemod	35
function::modname	36
function::symname	37
function::symdata	38
function::usymname	39
function::usymdata	40
function::print_ustack	41
function::sprint_ustack	42
function::print_backtrace	43
function::sprint_backtrace	44
function::backtrace	45
function::task_backtrace	46
function::caller	47
function::caller_addr	48
function::print_ubacktrace	49
function::sprint_ubacktrace	50
function::print_ubacktrace_brief	51
function::ubacktrace	52
function::task_current	53
function::task_parent	54
function::task_state	55

function::task_execname	56
function::task_pid	57
function::pid2task	58
function::pid2execname	59
function::task_tid	60
function::task_gid	61
function::task_egid	62
function::task_uid	63
function::task_euid	64
function::task_prio	65
function::task_nice	66
function::task_cpu	67
function::task_open_file_handles	68
function::task_max_file_handles	69
function::pn	70
3. Timestamp Functions	71
function::get_cycles	72
function::gettimeofday_ns	73
function::gettimeofday_us	74
function::gettimeofday_ms	75
function::gettimeofday_s	76
4. Time string utility function	77
function::ctime	78
5. Memory Tapset	79
function::vm_fault_contains	80
probe::vm.pagefault	81
probe::vm.pagefault.return	82
function::addr_to_node	83
probe::vm.write_shared	84
probe::vm.write_shared_copy	85
probe::vm.mmap	86
probe::vm.munmap	87
probe::vm.brk	88
probe::vm.oom_kill	89
probe::vm.kmalloc	90
probe::vm.kmem_cache_alloc	91
probe::vm.kmalloc_node	92
probe::vm.kmem_cache_alloc_node	93
probe::vm.kfree	94
probe::vm.kmem_cache_free	95
function::proc_mem_size	96
function::proc_mem_size_pid	97
function::proc_mem_rss	98
function::proc_mem_rss_pid	99
function::proc_mem_shr	100
function::proc_mem_shr_pid	101
function::proc_mem_txt	102
function::proc_mem_txt_pid	103
function::proc_mem_data	104
function::proc_mem_data_pid	105
function::mem_page_size	106
function::bytes_to_string	107
function::pages_to_string	108
function::proc_mem_string	109
function::proc_mem_string_pid	110
6. Task Time Tapset	111
function::task_utime	112
function::task_utime_tid	113

function::task_time	114
function::task_time_tid	115
function::cputime_to_msecs	116
function::msecs_to_string	117
function::cputime_to_string	118
function::task_time_string	119
function::task_time_string_tid	120
7. IO Scheduler and block IO Tapset	121
probe::ioscheduler.elv_next_request	122
probe::ioscheduler.elv_next_request.return	123
probe::ioscheduler.elv_completed_request	124
probe::ioscheduler.elv_add_request.kp	125
probe::ioscheduler.elv_add_request.tp	126
probe::ioscheduler.elv_add_request	127
probe::ioscheduler_trace.elv_completed_request	128
probe::ioscheduler_trace.elv_issue_request	129
probe::ioscheduler_trace.elv_requeue_request	130
probe::ioscheduler_trace.elv_abort_request	131
probe::ioscheduler_trace.plugin	132
probe::ioscheduler_trace.unplug_io	133
probe::ioscheduler_trace.unplug_timer	134
probe::ioblock.request	135
probe::ioblock.end	136
probe::ioblock_trace.bounce	137
probe::ioblock_trace.request	138
probe::ioblock_trace.end	139
8. SCSI Tapset	140
probe::scsi.ioentry	141
probe::scsi.iodispatching	142
probe::scsi.iodone	143
probe::scsi.iocompleted	144
probe::scsi.ioexecute	145
probe::scsi.set_state	146
9. TTY Tapset	147
probe::tty.open	148
probe::tty.release	149
probe::tty.resize	150
probe::tty.ioctl	151
probe::tty.init	152
probe::tty.register	153
probe::tty.unregister	154
probe::tty.poll	155
probe::tty.receive	156
probe::tty.write	157
probe::tty.read	158
10. Networking Tapset	159
probe::netdev.receive	160
probe::netdev.transmit	161
probe::netdev.change_mtu	162
probe::netdev.open	163
probe::netdev.close	164
probe::netdev.hard_transmit	165
probe::netdev.rx	166
probe::netdev.change_rx_flag	167
probe::netdev.set_promiscuity	168
probe::netdev.ioctl	169
probe::netdev.register	170
probe::netdev.unregister	171

probe::netdev.get_stats	172
probe::netdev.change_mac	173
probe::tcp.sendmsg	174
probe::tcp.sendmsg.return	175
probe::tcp.recvmsg	176
probe::tcp.recvmsg.return	177
probe::tcp.disconnect	178
probe::tcp.disconnect.return	179
probe::tcp.setsockopt	180
probe::tcp.setsockopt.return	181
probe::tcp.receive	182
probe::udp.sendmsg	183
probe::udp.sendmsg.return	184
probe::udp.recvmsg	185
probe::udp.recvmsg.return	186
probe::udp.disconnect	187
probe::udp.disconnect.return	188
function::ip_ntop	189
11. Socket Tapset	190
probe::socket.send	191
probe::socket.receive	192
probe::socket.sendmsg	193
probe::socket.sendmsg.return	194
probe::socket.recvmsg	195
probe::socket.recvmsg.return	196
probe::socket.aio_write	197
probe::socket.aio_write.return	198
probe::socket.aio_read	199
probe::socket.aio_read.return	200
probe::socket.writev	201
probe::socket.writev.return	202
probe::socket.readv	203
probe::socket.readv.return	204
probe::socket.create	205
probe::socket.create.return	206
probe::socket.close	207
probe::socket.close.return	208
function::sock_prot_num2str	209
function::sock_prot_str2num	210
function::sock_fam_num2str	211
function::sock_fam_str2num	212
function::sock_state_num2str	213
function::sock_state_str2num	214
12. Kernel Process Tapset	215
probe::kprocess.create	216
probe::kprocess.start	217
probe::kprocess.exec	218
probe::kprocess.exec_complete	219
probe::kprocess.exit	220
probe::kprocess.release	221
13. Signal Tapset	222
probe::signal.send	223
probe::signal.send.return	224
probe::signal.checkperm	225
probe::signal.checkperm.return	226
probe::signal.wakeup	227
probe::signal.check_ignored	228
probe::signal.check_ignored.return	229

probe::signal.force_segv	230
probe::signal.force_segv.return	231
probe::signal.syskill	232
probe::signal.syskill.return	233
probe::signal.sys_tkill	234
probe::signal.systkill.return	235
probe::signal.sys_tgkill	236
probe::signal.sys_tgkill.return	237
probe::signal.send_sig_queue	238
probe::signal.send_sig_queue.return	239
probe::signal.pending	240
probe::signal.pending.return	241
probe::signal.handle	242
probe::signal.handle.return	243
probe::signal.do_action	244
probe::signal.do_action.return	245
probe::signal.procmask	246
probe::signal.procmask.return	247
probe::signal.flush	248
14. Directory-entry (dentry) Tapset	249
function::d_name	250
function::reverse_path_walk	251
function::d_path	252
15. Logging Tapset	253
function::log	254
function::warn	255
function::exit	256
function::error	257
function::ftrace	258
16. Random functions Tapset	259
function::randint	260
17. String and data retrieving functions Tapset	261
function::kernel_string	262
function::kernel_string2	263
function::kernel_string_n	264
function::kernel_long	265
function::kernel_int	266
function::kernel_short	267
function::kernel_char	268
function::kernel_pointer	269
function::user_string	270
function::user_string2	271
function::user_string_warn	272
function::user_string_quoted	273
function::user_string_n	274
function::user_string_n2	275
function::user_string_n_warn	276
function::user_string_n_quoted	277
function::user_short	278
function::user_short_warn	279
function::user_int	280
function::user_int_warn	281
function::user_long	282
function::user_long_warn	283
function::user_char	284
function::user_char_warn	285
18. A collection of standard string functions	286
function::strlen	287

function::substr	288
function::stringat	289
function::isinstr	290
function::text_str	291
function::text_strn	292
function::tokenize	293
function::str_replace	294
function::strtol	295
function::isdigit	296
19. Utility functions for using ansi control chars in logs	297
function::ansi_clear_screen	298
function::ansi_set_color	299
function::ansi_set_color2	300
function::ansi_set_color3	301
function::ansi_reset_color	302
function::ansi_new_line	303
function::ansi_cursor_move	304
function::ansi_cursor_hide	305
function::ansi_cursor_save	306
function::ansi_cursor_restore	307
function::ansi_cursor_show	308

Chapter 1. Introduction

SystemTap provides free software (GPL) infrastructure to simplify the gathering of information about the running Linux system. This assists diagnosis of a performance or functional problem. SystemTap eliminates the need for the developer to go through the tedious and disruptive instrument, recompile, install, and reboot sequence that may be otherwise required to collect data.

SystemTap provides a simple command line interface and scripting language for writing instrumentation for a live running kernel. The instrumentation makes extensive use of the probe points and functions provided in the *tapset* library. This document describes the various probe points and functions.

Tapset Name Format

In this guide, tapset definitions appear in the following format:

```
name: return (parameters)
definition
```

The *return* field specifies what data type the tapset extracts and returns from the kernel during a probe (and thus, returns). Tapsets use 2 data types for *return*: *long* (tapset extracts and returns an integer) and *string* (tapset extracts and returns a string).

In some cases, tapsets do not have a *return* value. This simply means that the tapset does not extract anything from the kernel. This is common among asynchronous events such as timers, exit functions, and print functions.

Chapter 2. Context Functions

The context functions provide additional information about where an event occurred. These functions can provide information such as a backtrace to where the event occurred and the current register values for the processor.

Name

function::print_regs — Print a register dump.

Synopsis

```
print_regs()
```

Arguments

None

General Syntax

```
print_regs
```

Description

This function prints a register dump.

Name

`function::execname` — Returns the `execname` of a target process (or group of processes).

Synopsis

```
execname:string()
```

Arguments

None

General Syntax

```
execname:string
```

Description

Returns the `execname` of a target process (or group of processes).

Name

function::pid — Returns the ID of a target process.

Synopsis

```
pid:long()
```

Arguments

None

General Syntax

```
pid:long
```

Description

This function returns the ID of a target process.

Name

function::tid — Returns the thread ID of a target process.

Synopsis

```
tid:long()
```

Arguments

None

General Syntax

```
tid:long
```

Description

This function returns the thread ID of the target process.

Name

function::ppid — Returns the process ID of a target process's parent process.

Synopsis

```
ppid:long()
```

Arguments

None

General Syntax

```
ppid:long
```

Description

This function return the process ID of the target proccess's parent process.

Name

function::pgrp — Returns the process group ID of the current process.

Synopsis

```
pgrp:long()
```

Arguments

None

General Syntax

```
pgrp:long
```

Description

This function returns the process group ID of the current process.

Name

function::sid — Returns the session ID of the current process.

Synopsis

```
sid:long()
```

Arguments

None

General Syntax

```
sid:long
```

Description

The session ID of a process is the process group ID of the session leader. Session ID is stored in the `signal_struct` since Kernel 2.6.0.

Name

`function::pexecname` — Returns the execname of a target process's parent process.

Synopsis

```
pexecname:string()
```

Arguments

None

General Syntax

```
pexecname:string
```

Description

This function returns the execname of a target process's parent process.

Name

`function::gid` — Returns the group ID of a target process.

Synopsis

```
gid:long()
```

Arguments

None

General Syntax

```
gid:long
```

Description

This function returns the group ID of a target process.

Name

function::egid — Returns the effective gid of a target process.

Synopsis

```
egid:long()
```

Arguments

None

General Syntax

```
egid:long
```

Description

This function returns the effective gid of a target process

Name

function::uid — Returns the user ID of a target process.

Synopsis

```
uid:long()
```

Arguments

None

General Syntax

```
uid:long
```

Description

This function returns the user ID of the target process.

Name

`function::euid` — Return the effective uid of a target process.

Synopsis

```
euid:long()
```

Arguments

None

General Syntax

```
euid:long
```

Description

Returns the effective user ID of the target process.

Name

function::is_myproc — Determines if the current probe point has occurred in the user's own process.

Synopsis

```
is_myproc:long()
```

Arguments

None

General Syntax

```
is_myproc:long
```

Description

This function returns 1 if the current probe point has occurred in the user's own process.

Name

function::cpu — Returns the current cpu number.

Synopsis

```
cpu:long()
```

Arguments

None

General Syntax

```
cpu:long
```

Description

This function returns the current cpu number.

Name

function::pp — Returns the active probe point.

Synopsis

```
pp:string()
```

Arguments

None

General Syntax

```
pp:string
```

Description

This function returns the fully-resolved probe point associated with a currently running probe handler, including alias and wild-card expansion effects. Context: The current probe point.

Name

`function::registers_valid` — Determines validity of `register` and `u_register` in current context.

Synopsis

```
registers_valid:long()
```

Arguments

None

General Syntax

```
registers_valid:long
```

Description

This function returns 1 if `register` and `u_register` can be used in the current context, or 0 otherwise. For example, `registers_valid` returns 0 when called from a begin or end probe.

Name

function::user_mode — Determines if probe point occurs in user-mode.

Synopsis

```
user_mode:long()
```

Arguments

None

General Syntax

```
user_mode:long
```

Return 1 if the probe point occurred in user-mode.

Name

function::is_return — Whether the current probe context is a return probe.

Synopsis

```
is_return:long()
```

Arguments

None

General Syntax

```
is_return:long
```

Description

Returns 1 if the current probe context is a return probe, returns 0 otherwise.

Name

`function::target` — Return the process ID of the target process.

Synopsis

```
target:long()
```

Arguments

None

General Syntax

```
target:long
```

Description

This function returns the process ID of the target process. This is useful in conjunction with the `-x` PID or `-c` CMD command-line options to `stap`. An example of its use is to create scripts that filter on a specific process.

`-x <pid> target` returns the pid specified by `-x -c <command> target` returns the pid for the executed command specified by `-c`

Name

`function::module_name` — The module name of the current script.

Synopsis

```
module_name:string()
```

Arguments

None

General Syntax

```
module_name:string
```

Description

This function returns the name of the stap module. Either generated randomly (`stap_[0-9a-f]+_[0-9a-f]+`) or set by `stap -m <module_name>`.

Name

function::stp_pid — The process id of the stapio process.

Synopsis

```
stp_pid:long()
```

Arguments

None

General Syntax

```
stp_pid:long
```

Description

This function returns the process id of the stapio process that launched this script. There could be other SystemTap scripts and stapio processes running on the system.

Name

`function::stack_size` — Return the size of the kernel stack.

Synopsis

```
stack_size:long()
```

Arguments

None

General Syntax

```
stack_size:long
```

Description

This function returns the size of the kernel stack.

Name

`function::stack_used` — Returns the amount of kernel stack used.

Synopsis

```
stack_used:long()
```

Arguments

None

General Syntax

```
stack_used:long
```

Description

This function determines how many bytes are currently used in the kernel stack.

Name

`function::stack_unused` — Returns the amount of kernel stack currently available.

Synopsis

```
stack_unused:long()
```

Arguments

None

General Syntax

```
stack_unused:long
```

Description

This function determines how many bytes are currently available in the kernel stack.

Name

function::uaddr — User space address of current running task. EXPERIMENTAL.

Synopsis

```
uaddr:long()
```

Arguments

None

General Syntax

```
uaddr:long
```

Description

Returns the address in userspace that the current task was at when the probe occurred. When the current running task isn't a user space thread, or the address cannot be found, zero is returned. Can be used to see where the current task is combined with `usymname` or `syndata`. Often the task will be in the VDSO where it entered the kernel. FIXME - need VDSO tracking support #10080.

Name

`function::cmdline_args` — Fetch command line arguments from current process

Synopsis

```
cmdline_args:string(n:long,m:long,delim:string)
```

Arguments

<i>n</i>	First argument to get (zero is the command itself)
<i>m</i>	Last argument to get (or minus one for all arguments after n)
<i>delim</i>	String to use to delimit arguments when more than one.

General Syntax

```
cmdline_args:string(n:long, m:long, delim:string)
```

Description

Returns arguments from the current process starting with argument number *n*, up to argument *m*. If there are less than *n* arguments, or the arguments cannot be retrieved from the current process, the empty string is returned. If *m* is smaller than *n* then all arguments starting from argument *n* are returned. Argument zero is traditionally the command itself.

Name

`function::cmdline_arg` — Fetch a command line argument.

Synopsis

```
cmdline_arg:string(n:long)
```

Arguments

n Argument to get (zero is the command itself)

General Syntax

```
cmdline_arg:string(n:long)
```

Description

Returns argument the requested argument from the current process or the empty string when there are not that many arguments or there is a problem retrieving the argument. Argument zero is traditionally the command itself.

Name

function::cmdline_str — Fetch all command line arguments from current process

Synopsis

```
cmdline_str:string()
```

Arguments

None

General Syntax

```
cmdline_str:string
```

Description

Returns all arguments from the current process delimited by spaces. Returns the empty string when the arguments cannot be retrieved.

Name

`function::env_var` — Fetch environment variable from current process

Synopsis

```
env_var:string(name:string)
```

Arguments

name Name of the environment variable to fetch

General Syntax

```
env_var:string(name:string)
```

Description

Returns the contents of the specified environment value for the current process. If the variable isn't set an empty string is returned.

Name

`function::print_stack` — Print out kernel stack from string.

Synopsis

```
print_stack(stk:string)
```

Arguments

stk String with list of hexadecimal addresses.

General Syntax

```
print_stack(stk:string)
```

Description

This function performs a symbolic lookup of the addresses in the given `string`, which is assumed to be the result of a prior call to `backtrace`.

Print one line per address, including the address, the name of the function containing the address, and an estimate of its position within that function. Return nothing.

Name

`function::sprint_stack` — Return stack for kernel addresses from string. EXPERIMENTAL!

Synopsis

```
sprint_stack:string(stk:string)
```

Arguments

stk String with list of hexadecimal (kernel) addresses.

Description

Perform a symbolic lookup of the addresses in the given string, which is assumed to be the result of a prior call to `backtrace`.

Returns a simple backtrace from the given hex string. One line per address. Includes the symbol name (or hex address if symbol couldn't be resolved) and module name (if found). Includes the offset from the start of the function if found, otherwise the offset will be added to the module (if found, between brackets). Returns the backtrace as string (each line terminated by a newline character). Note that the returned stack will be truncated to `MAXSTRINGLEN`, to print fuller and richer stacks use `print_stack`.

Name

`function::probefunc` — Return the probe point's function name, if known.

Synopsis

```
probefunc:string()
```

Arguments

None

General Syntax

```
probefunc:string
```

Description

This function returns the name of the function being probed. It will do this based on the probe point string as returned by `pp`.

Please note

this function is deprecated, please use `symname` and/or `usymname`. This function might return a function name based on the current address if the probe point context couldn't be parsed.

Name

function::probemod — Return the probe point's kernel module name.

Synopsis

```
probemod:string()
```

Arguments

None

General Syntax

```
probemod:string
```

Description

This function returns the name of the kernel module containing the probe point, if known.

Name

`function::modname` — Return the kernel module name loaded at the address.

Synopsis

```
modname:string(addr:long)
```

Arguments

addr The address.

Description

Returns the module name associated with the given address if known. If not known it will return the string “<unknown>”. If the address was not in a kernel module, but in the kernel itself, then the string “kernel” will be returned.

Name

`function::symname` — Return the kernel symbol associated with the given address.

Synopsis

```
symname:string(addr:long)
```

Arguments

addr The address to translate.

General Syntax

```
symname:string(addr:long)
```

Description

Returns the (function) symbol name associated with the given address if known. If not known it will return the hex string representation of `addr`.

Name

`function::symdata` — Return the kernel symbol and module offset for the address.

Synopsis

```
symdata:string(addr:long)
```

Arguments

addr The address to translate.

General Syntax

```
symdata:string(addr:long)
```

Description

Returns the (function) symbol name associated with the given address if known, the offset from the start and size of the symbol, plus module name (between brackets). If symbol is unknown, but module is known, the offset inside the module, plus the size of the module is added. If any element is not known it will be omitted and if the symbol name is unknown it will return the hex string for the given address.

Name

`function::usymname` — Return the symbol of an address in the current task. EXPERIMENTAL!

Synopsis

```
usymname:string(addr:long)
```

Arguments

addr The address to translate.

Description

Returns the (function) symbol name associated with the given address if known. If not known it will return the hex string representation of `addr`.

Name

`function::usymdata` — Return the symbol and module offset of an address. EXPERIMENTAL!

Synopsis

```
usymdata:string(addr:long)
```

Arguments

addr The address to translate.

Description

Returns the (function) symbol name associated with the given address in the current task if known, the offset from the start and the size of the symbol, plus the module name (between brackets). If symbol is unknown, but module is known, the offset inside the module, plus the size of the module is added. If any element is not known it will be omitted and if the symbol name is unknown it will return the hex string for the given address.

Name

`function::print_ustack` — Print out stack for the current task from string. EXPERIMENTAL!

Synopsis

```
print_ustack(stk:string)
```

Arguments

stk String with list of hexadecimal addresses for the current task.

Description

Perform a symbolic lookup of the addresses in the given string, which is assumed to be the result of a prior call to `ubacktrace` for the current task.

Print one line per address, including the address, the name of the function containing the address, and an estimate of its position within that function. Return nothing.

Name

`function::sprint_ustack` — Return stack for the current task from string. EXPERIMENTAL!

Synopsis

```
sprint_ustack:string(stk:string)
```

Arguments

stk String with list of hexadecimal addresses for the current task.

Description

Perform a symbolic lookup of the addresses in the given string, which is assumed to be the result of a prior call to `ubacktrace` for the current task.

Returns a simple backtrace from the given hex string. One line per address. Includes the symbol name (or hex address if symbol couldn't be resolved) and module name (if found). Includes the offset from the start of the function if found, otherwise the offset will be added to the module (if found, between brackets). Returns the backtrace as string (each line terminated by a newline character). Note that the returned stack will be truncated to `MAXSTRINGLEN`, to print fuller and richer stacks use `print_ustack`.

Name

function::print_backtrace — Print stack back trace

Synopsis

```
print_backtrace()
```

Arguments

None

General Syntax

```
print_backtrace
```

Description

This function is equivalent to `print_stack(backtrace)`, except that deeper stack nesting may be supported. The function does not return a value.

Name

function::sprint_backtrace — Return stack back trace as string. EXPERIMENTAL!

Synopsis

```
sprint_backtrace()
```

Arguments

None

Description

Returns a simple (kernel) backtrace. One line per address. Includes the symbol name (or hex address if symbol couldn't be resolved) and module name (if found). Includes the offset from the start of the function if found, otherwise the offset will be added to the module (if found, between brackets). Returns the backtrace as string (each line terminated by a newline character). Note that the returned stack will be truncated to MAXSTRINGLEN, to print fuller and richer stacks use `print_backtrace`. Equivalent to `sprint_stack(backtrace)`, but more efficient (no need to translate between hex strings and final backtrace string).

Name

`function::backtrace` — Hex backtrace of current stack

Synopsis

```
backtrace:string()
```

Arguments

None

General Syntax

```
backtrace:string
```

Description

This function returns a string of hex addresses that are a backtrace of the stack. Output may be truncated as as per maximum string length (MAXSTRINGLEN).

Name

function::task_backtrace — Hex backtrace of an arbitrary task

Synopsis

```
task_backtrace:string(task:long)
```

Arguments

task pointer to task_struct

General Syntax

```
task_backtrace:string(task:long)
```

Description

This function returns a string of hex addresses that are a backtrace of the stack of a particular task. Output may be truncated as per maximum string length.

Name

function::caller — Return name and address of calling function

Synopsis

```
caller:string()
```

Arguments

None

General Syntax

```
caller:string
```

Description

This function returns the address and name of the calling function. This is equivalent to calling: `sprintf("s 0xx", symname(caller_addr, caller_addr))` Works only for return probes at this time.

Name

`function::caller_addr` — Return caller address

Synopsis

```
caller_addr:long()
```

Arguments

None

General Syntax

```
caller_addr:long
```

Description

This function returns the address of the calling function. Works only for return probes at this time.

Name

`function::print_ubacktrace` — Print stack back trace for current task. EXPERIMENTAL!

Synopsis

```
print_ubacktrace()
```

Arguments

None

Description

Equivalent to `print_ustack(ubacktrace)`, except that deeper stack nesting may be supported.
Return nothing.

Name

function::sprint_ubacktrace — Return stack back trace for current task as string. EXPERIMENTAL!

Synopsis

```
sprint_ubacktrace()
```

Arguments

None

Description

Returns a simple backtrace for the current task. One line per address. Includes the symbol name (or hex address if symbol couldn't be resolved) and module name (if found). Includes the offset from the start of the function if found, otherwise the offset will be added to the module (if found, between brackets). Returns the backtrace as string (each line terminated by a newline character). Note that the returned stack will be truncated to MAXSTRINGLEN, to print fuller and richer stacks use `print_ubacktrace`. Equivalent to `sprint_ustack(ubacktrace)`, but more efficient (no need to translate between hex strings and final backtrace string).

Name

function::print_ubacktrace_brief — Print stack back trace for current task. EXPERIMENTAL!

Synopsis

```
print_ubacktrace_brief()
```

Arguments

None

Description

Equivalent to `print_ubacktrace`, but output for each symbol is shorter (just name and offset, or just the hex address of no symbol could be found).

Name

function::ubacktrace — Hex backtrace of current task stack. EXPERIMENTAL!

Synopsis

```
ubacktrace:string()
```

Arguments

None

Description

Return a string of hex addresses that are a backtrace of the stack of the current task. Output may be truncated as per maximum string length. Returns empty string when current probe point cannot determine user backtrace.

Name

function::task_current — The current task_struct of the current task.

Synopsis

```
task_current:long()
```

Arguments

None

General Syntax

```
task_current:long
```

Description

This function returns the task_struct representing the current process. This address can be passed to the various task_*() functions to extract more task-specific data.

Name

`function::task_parent` — The `task_struct` of the parent task.

Synopsis

```
task_parent:long(task:long)
```

Arguments

task `task_struct` pointer.

General Syntax

```
task_parent:long(task:long)
```

Description

This function returns the parent `task_struct` of the given task. This address can be passed to the various `task_*`() functions to extract more task-specific data.

Name

`function::task_state` — The state of the task.

Synopsis

```
task_state:long(task:long)
```

Arguments

task `task_struct` pointer.

General Syntax

```
task_state:long(task:long)
```

Description

Return the state of the given task, one of: `TASK_RUNNING` (0), `TASK_INTERRUPTIBLE` (1), `TASK_UNINTERRUPTIBLE` (2), `TASK_STOPPED` (4), `TASK_TRACED` (8), `EXIT_ZOMBIE` (16), `EXIT_DEAD` (32).

Name

`function::task_execname` — The name of the task.

Synopsis

```
task_execname:string(task:long)
```

Arguments

task task_struct pointer.

General Syntax

```
task_execname:string(task:long)
```

Description

Return the name of the given task.

Name

`function::task_pid` — The process identifier of the task.

Synopsis

```
task_pid:long (task:long)
```

Arguments

task `task_struct` pointer.

General Syntax

```
task_pid:long (task:long)
```

Description

This fuction returns the process id of the given task.

Name

function::pid2task — The task_struct of the given process identifier.

Synopsis

```
pid2task:long(pid:long)
```

Arguments

pid Process identifier.

Description

Return the task struct of the given process id.

Name

function::pid2execname — The name of the given process identifier.

Synopsis

```
pid2execname:string(pid:long)
```

Arguments

pid Process identifier.

Description

Return the name of the given process id.

Name

function::task_tid — The thread identifier of the task.

Synopsis

```
task_tid:long(task:long)
```

Arguments

task task_struct pointer.

General Syntax

```
task_tid:long(task:long)
```

Description

This function returns the thread id of the given task.

Name

function::task_gid — The group identifier of the task.

Synopsis

```
task_gid:long(task:long)
```

Arguments

task task_struct pointer.

General Syntax

```
task_gid:long(task:long)
```

Description

This function returns the group id of the given task.

Name

`function::task_egid` — The effective group identifier of the task.

Synopsis

```
task_egid:long(task:long)
```

Arguments

task task_struct pointer.

General Syntax

```
task_egid:long(task:long)
```

Description

This function returns the effective group id of the given task.

Name

function::task_uid — The user identifier of the task.

Synopsis

```
task_uid:long(task:long)
```

Arguments

task task_struct pointer.

General Syntax

```
task_uid:long(task:long)
```

Description

This function returns the user id of the given task.

Name

`function::task_euid` — The effective user identifier of the task.

Synopsis

```
task_euid:long(task:long)
```

Arguments

task task_struct pointer.

General Syntax

```
task_euid:long(task:long)
```

Description

This function returns the effective user id of the given task.

Name

function::task_prio — The priority value of the task.

Synopsis

```
task_prio:long(task:long)
```

Arguments

task task_struct pointer.

General Syntax

```
task_prio:long(task:long)
```

Description

This function returns the priority value of the given task.

Name

function::task_nice — The nice value of the task.

Synopsis

```
task_nice:long(task:long)
```

Arguments

task task_struct pointer.

General Syntax

```
task_nice:long(task:long)
```

Description

This function returns the nice value of the given task.

Name

function::task_cpu — The scheduled cpu of the task.

Synopsis

```
task_cpu:long(task:long)
```

Arguments

task task_struct pointer.

General Syntax

```
task_cpu:long(task:long)
```

Description

This function returns the scheduled cpu for the given task.

Name

`function::task_open_file_handles` — The number of open files of the task.

Synopsis

```
task_open_file_handles:long(task:long)
```

Arguments

task task_struct pointer.

General Syntax

```
task_open_file_handles:long(task:long)
```

Description

This function returns the number of open file handlers for the given task.

Name

function::task_max_file_handles — The max number of open files for the task.

Synopsis

```
task_max_file_handles:long(task:long)
```

Arguments

task task_struct pointer.

General Syntax

```
task_max_file_handles:long(task:long)
```

Description

This function returns the maximum number of file handlers for the given task.

Name

function::pn — Returns the active probe name.

Synopsis

```
pn:string()
```

Arguments

None

General Syntax

```
pn:string
```

Description

This function returns the script-level probe point associated with a currently running probe handler, including wild-card expansion effects. Context: The current probe point.

Chapter 3. Timestamp Functions

Each timestamp function returns a value to indicate when a function is executed. These returned values can then be used to indicate when an event occurred, provide an ordering for events, or compute the amount of time elapsed between two time stamps.

Name

function::get_cycles — Processor cycle count.

Synopsis

```
get_cycles:long()
```

Arguments

None

General Syntax

```
get_cycles:long
```

Description

This function returns the processor cycle counter value if available, else it returns zero. The cycle counter is free running and unsynchronized on each processor. Thus, the order of events cannot be determined by comparing the results of the `get_cycles` function on different processors.

Name

function::gettimeofday_ns — Number of nanoseconds since UNIX epoch.

Synopsis

```
gettimeofday_ns:long()
```

Arguments

None

General Syntax

```
gettimeofday_ns:long
```

Description

This function returns the number of nanoseconds since the UNIX epoch.

Name

function::gettimeofday_us — Number of microseconds since UNIX epoch.

Synopsis

```
gettimeofday_us:long()
```

Arguments

None

General Syntax

```
gettimeofday_us:long
```

Description

This function returns the number of microseconds since the UNIX epoch.

Name

function::gettimeofday_ms — Number of milliseconds since UNIX epoch.

Synopsis

```
gettimeofday_ms:long()
```

Arguments

None

General Syntax

```
gettimeofday_ms:long
```

Description

This function returns the number of milliseconds since the UNIX epoch.

Name

function::gettimeofday_s — Number of seconds since UNIX epoch.

Synopsis

```
gettimeofday_s:long()
```

Arguments

None

General Syntax

```
gettimeofday_s:long
```

Description

This function returns the number of seconds since the UNIX epoch.

Chapter 4. Time string utility function

Utility function to turn seconds since the epoch (as returned by the timestamp function `gettimeofday_s()`) into a human readable date/time string.

Name

`function::ctime` — Convert seconds since epoch into human readable date/time string.

Synopsis

```
ctime:string(epochsecs:long)
```

Arguments

epochsecs Number of seconds since epoch (as returned by `gettimeofday_s`).

General Syntax

```
ctime:string(epochsecs:long)
```

Description

Takes an argument of seconds since the epoch as returned by `gettimeofday_s`. Returns a string of the form

```
“Wed Jun 30 21:49:08 1993”
```

The string will always be exactly 24 characters. If the time would be unreasonable far in the past (before what can be represented with a 32 bit offset in seconds from the epoch) the returned string will be “a long, long time ago...”. If the time would be unreasonable far in the future the returned string will be “far far in the future...” (both these strings are also 24 characters wide).

Note that the epoch (zero) corresponds to

```
“Thu Jan 1 00:00:00 1970”
```

The earliest full date given by `ctime`, corresponding to `epochsecs -2147483648` is “Fri Dec 13 20:45:52 1901”. The latest full date given by `ctime`, corresponding to `epochsecs 2147483647` is “Tue Jan 19 03:14:07 2038”.

The abbreviations for the days of the week are ‘Sun’, ‘Mon’, ‘Tue’, ‘Wed’, ‘Thu’, ‘Fri’, and ‘Sat’. The abbreviations for the months are ‘Jan’, ‘Feb’, ‘Mar’, ‘Apr’, ‘May’, ‘Jun’, ‘Jul’, ‘Aug’, ‘Sep’, ‘Oct’, ‘Nov’, and ‘Dec’.

Note that the real C library `ctime` function puts a newline (‘\n’) character at the end of the string that this function does not. Also note that since the kernel has no concept of timezones, the returned time is always in GMT.

Chapter 5. Memory Tapset

This family of probe points is used to probe memory-related events or query the memory usage of the current process. It contains the following probe points:

Name

function::vm_fault_contains — Test return value for page fault reason

Synopsis

```
vm_fault_contains:long (value:long, test:long)
```

Arguments

<i>value</i>	The fault_type returned by vm.page_fault.return
<i>test</i>	The type of fault to test for (VM_FAULT_OOM or similar)

Name

probe::vm.pagefault — Records that a page fault occurred.

Synopsis

```
vm.pagefault
```

Values

write_access Indicates whether this was a write or read access; 1 indicates a write, while 0 indicates a read.

name Name of the probe point

address The address of the faulting memory access; i.e. the address that caused the page fault.

Context

The process which triggered the fault

Name

probe::vm.pagefault.return — Indicates what type of fault occurred.

Synopsis

```
vm.pagefault.return
```

Values

<i>name</i>	Name of the probe point
<i>fault_type</i>	Returns either 0 (VM_FAULT_OOM) for out of memory faults, 2 (VM_FAULT_MINOR) for minor faults, 3 (VM_FAULT_MAJOR) for major faults, or 1 (VM_FAULT_SIGBUS) if the fault was neither OOM, minor fault, nor major fault.

Name

`function::addr_to_node` — Returns which node a given address belongs to within a NUMA system.

Synopsis

```
addr_to_node:long(addr:long)
```

Arguments

addr The address of the faulting memory access.

General Syntax

```
addr_to_node:long(addr:long)
```

Description

This function accepts an address, and returns the node that the given address belongs to in a NUMA system.

Name

probe::vm.write_shared — Attempts at writing to a shared page.

Synopsis

```
vm.write_shared
```

Values

<i>name</i>	Name of the probe point
<i>address</i>	The address of the shared write.

Context

The context is the process attempting the write.

Description

Fires when a process attempts to write to a shared page. If a copy is necessary, this will be followed by a `vm.write_shared_copy`.

Name

probe::vm.write_shared_copy — Page copy for shared page write.

Synopsis

```
vm.write_shared_copy
```

Values

<i>name</i>	Name of the probe point
<i>zero</i>	Boolean indicating whether it is a zero page (can do a clear instead of a copy).
<i>address</i>	The address of the shared write.

Context

The process attempting the write.

Description

Fires when a write to a shared page requires a page copy. This is always preceded by a `vm.shared_write`.

Name

probe::vm.mmap — Fires when an mmap is requested.

Synopsis

`vm.mmap`

Values

<i>length</i>	The length of the memory segment
<i>name</i>	Name of the probe point
<i>address</i>	The requested address

Context

The process calling mmap.

Name

probe::vm.munmap — Fires when an munmap is requested.

Synopsis

`vm.munmap`

Values

<i>length</i>	The length of the memory segment
<i>name</i>	Name of the probe point
<i>address</i>	The requested address

Context

The process calling munmap.

Name

probe::vm.brk — Fires when a brk is requested (i.e. the heap will be resized).

Synopsis

```
vm.brk
```

Values

<i>length</i>	The length of the memory segment
<i>name</i>	Name of the probe point
<i>address</i>	The requested address

Context

The process calling brk.

Name

probe::vm.oom_kill — Fires when a thread is selected for termination by the OOM killer.

Synopsis

```
vm.oom_kill
```

Values

name Name of the probe point

task The task being killed

Context

The process that tried to consume excessive memory, and thus triggered the OOM.

Name

probe::vm.kmalloc — Fires when kmalloc is requested.

Synopsis

```
vm.kmalloc
```

Values

<i>ptr</i>	Pointer to the kmemory allocated
<i>caller_function</i>	Name of the caller function.
<i>call_site</i>	Address of the kmemory function.
<i>gfp_flag_name</i>	type of kmemory to allocate (in String format)
<i>name</i>	Name of the probe point
<i>bytes_req</i>	Requested Bytes
<i>bytes_alloc</i>	Allocated Bytes
<i>gfp_flags</i>	type of kmemory to allocate

Name

probe::vm.kmem_cache_alloc — Fires when \

Synopsis

`vm.kmem_cache_alloc`

Values

<i>ptr</i>	Pointer to the kmemory allocated
<i>caller_function</i>	Name of the caller function.
<i>call_site</i>	Address of the function calling this kmemory function.
<i>gfp_flag_name</i>	Type of kmemory to allocate(in string format)
<i>name</i>	Name of the probe point
<i>bytes_req</i>	Requested Bytes
<i>bytes_alloc</i>	Allocated Bytes
<i>gfp_flags</i>	type of kmemory to allocate

Description

`kmem_cache_alloc` is requested.

Name

probe::vm.kmalloc_node — Fires when kmalloc_node is requested.

Synopsis

```
vm.kmalloc_node
```

Values

<i>ptr</i>	Pointer to the kmemory allocated
<i>caller_function</i>	Name of the caller function.
<i>call_site</i>	Address of the function caling this kmemory function.
<i>gfp_flag_name</i>	Type of kmemory to allocate(in string format)
<i>name</i>	Name of the probe point
<i>bytes_req</i>	Requested Bytes
<i>bytes_alloc</i>	Allocated Bytes
<i>gfp_flags</i>	type of kmemory to allocate

Name

probe::vm.kmem_cache_alloc_node — Fires when \

Synopsis

vm.kmem_cache_alloc_node

Values

<i>ptr</i>	Pointer to the kmemory allocated
<i>caller_function</i>	Name of the caller function.
<i>call_site</i>	Address of the function calling this kmemory function.
<i>gfp_flag_name</i>	Type of kmemory to allocate(in string format)
<i>name</i>	Name of the probe point
<i>bytes_req</i>	Requested Bytes
<i>bytes_alloc</i>	Allocated Bytes
<i>gfp_flags</i>	type of kmemory to allocate

Description

kmem_cache_alloc_node is requested.

Name

probe::vm.kfree — Fires when kfree is requested.

Synopsis

```
vm.kfree
```

Values

<i>ptr</i>	Pointer to the kmemory allocated which is returned by kmalloc
<i>caller_function</i>	Name of the caller function.
<i>call_site</i>	Address of the function calling this kmemory function.
<i>name</i>	Name of the probe point

Name

probe::vm.kmem_cache_free — Fires when \

Synopsis

vm.kmem_cache_free

Values

<i>ptr</i>	Pointer to the kmemory allocated which is returned by kmem_cache
<i>caller_function</i>	Name of the caller function.
<i>call_site</i>	Address of the function calling this kmemory function.
<i>name</i>	Name of the probe point

Description

kmem_cache_free is requested.

Name

function::proc_mem_size — Total program virtual memory size in pages

Synopsis

```
proc_mem_size:long()
```

Arguments

None

Description

Returns the total virtual memory size in pages of the current process, or zero when there is no current process or the number of pages couldn't be retrieved.

Name

function::proc_mem_size_pid — Total program virtual memory size in pages

Synopsis

```
proc_mem_size_pid:long (pid:long)
```

Arguments

pid The pid of process to examine

Description

Returns the total virtual memory size in pages of the given process, or zero when that process doesn't exist or the number of pages couldn't be retrieved.

Name

function::proc_mem_rss — Program resident set size in pages

Synopsis

```
proc_mem_rss:long()
```

Arguments

None

Description

Returns the resident set size in pages of the current process, or zero when there is no current process or the number of pages couldn't be retrieved.

Name

function::proc_mem_rss_pid — Program resident set size in pages

Synopsis

```
proc_mem_rss_pid:long(pid:long)
```

Arguments

pid The pid of process to examine

Description

Returns the resident set size in pages of the given process, or zero when the process doesn't exist or the number of pages couldn't be retrieved.

Name

function::proc_mem_shr — Program shared pages (from shared mappings)

Synopsis

```
proc_mem_shr:long()
```

Arguments

None

Description

Returns the shared pages (from shared mappings) of the current process, or zero when there is no current process or the number of pages couldn't be retrieved.

Name

function::proc_mem_shr_pid — Program shared pages (from shared mappings)

Synopsis

```
proc_mem_shr_pid:long(pid:long)
```

Arguments

pid The pid of process to examine

Description

Returns the shared pages (from shared mappings) of the given process, or zero when the process doesn't exist or the number of pages couldn't be retrieved.

Name

function::proc_mem_txt — Program text (code) size in pages

Synopsis

```
proc_mem_txt:long()
```

Arguments

None

Description

Returns the current process text (code) size in pages, or zero when there is no current process or the number of pages couldn't be retrieved.

Name

function::proc_mem_txt_pid — Program text (code) size in pages

Synopsis

```
proc_mem_txt_pid:long(pid:long)
```

Arguments

pid The pid of process to examine

Description

Returns the given process text (code) size in pages, or zero when the process doesn't exist or the number of pages couldn't be retrieved.

Name

function::proc_mem_data — Program data size (data + stack) in pages

Synopsis

```
proc_mem_data:long()
```

Arguments

None

Description

Returns the current process data size (data + stack) in pages, or zero when there is no current process or the number of pages couldn't be retrieved.

Name

function::proc_mem_data_pid — Program data size (data + stack) in pages

Synopsis

```
proc_mem_data_pid:long(pid:long)
```

Arguments

pid The pid of process to examine

Description

Returns the given process data size (data + stack) in pages, or zero when the process doesn't exist or the number of pages couldn't be retrieved.

Name

function::mem_page_size — Number of bytes in a page for this architecture

Synopsis

```
mem_page_size:long()
```

Arguments

None

Name

function::bytes_to_string — Human readable string for given bytes

Synopsis

```
bytes_to_string:string(bytes:long)
```

Arguments

bytes Number of bytes to translate.

Description

Returns a string representing the number of bytes (up to 1024 bytes), the number of kilobytes (when less than 1024K) postfixed by 'K', the number of megabytes (when less than 1024M) postfixed by 'M' or the number of gigabytes postfixed by 'G'. If representing K, M or G, and the number is amount is less than 100, it includes a '.' plus the remainder. The returned string will be 5 characters wide (padding with whitespace at the front) unless negative or representing more than 9999G bytes.

Name

`function::pages_to_string` — Turns pages into a human readable string

Synopsis

```
pages_to_string:string(pages:long)
```

Arguments

pages Number of pages to translate.

Description

Multiplies `pages` by `page_size` to get the number of bytes and returns the result of `bytes_to_string`.

Name

function::proc_mem_string — Human readable string of current proc memory usage

Synopsis

```
proc_mem_string:string()
```

Arguments

None

Description

Returns a human readable string showing the size, rss, shr, txt and data of the memory used by the current process. For example “size: 301m, rss: 11m, shr: 8m, txt: 52k, data: 2248k”.

Name

function::proc_mem_string_pid — Human readable string of process memory usage

Synopsis

```
proc_mem_string_pid:string(pid:long)
```

Arguments

pid The pid of process to examine

Description

Returns a human readable string showing the size, rss, shr, txt and data of the memory used by the given process. For example “size: 301m, rss: 11m, shr: 8m, txt: 52k, data: 2248k”.

Chapter 6. Task Time Tapset

This tapset defines utility functions to query time related properties of the current tasks, translate those in milliseconds and human readable strings.

Name

function::task_etime — User time of the current task

Synopsis

```
task_etime:long()
```

Arguments

None

Description

Returns the user time of the current task in cputime. Does not include any time used by other tasks in this process, nor does it include any time of the children of this task.

Name

`function::task_utime_tid` — User time of the given task

Synopsis

```
task_utime_tid:long (tid:long)
```

Arguments

tid Thread id of the given task

Description

Returns the user time of the given task in cputime, or zero if the task doesn't exist. Does not include any time used by other tasks in this process, nor does it include any time of the children of this task.

Name

function::task_time — System time of the current task

Synopsis

```
task_time:long()
```

Arguments

None

Description

Returns the system time of the current task in cputime. Does not include any time used by other tasks in this process, nor does it include any time of the children of this task.

Name

`function::task_time_tid` — System time of the given task

Synopsis

```
task_time_tid:long (tid:long)
```

Arguments

tid Thread id of the given task

Description

Returns the system time of the given task in cputime, or zero if the task doesn't exist. Does not include any time used by other tasks in this process, nor does it include any time of the children of this task.

Name

function::cputime_to_msecs — Translates the given cputime into milliseconds

Synopsis

```
cputime_to_msecs:long (cputime:long)
```

Arguments

cputime Time to convert to milliseconds.

Name

`function::msecs_to_string` — Human readable string for given milliseconds

Synopsis

```
msecs_to_string:string(msecs:long)
```

Arguments

msecs Number of milliseconds to translate.

Description

Returns a string representing the number of milliseconds as a human readable string consisting of “XmY.ZZZs”, where X is the number of minutes, Y is the number of seconds and ZZZ is the number of milliseconds.

Name

function::cputime_to_string — Human readable string for given cputime

Synopsis

```
cputime_to_string:string(cputime:long)
```

Arguments

cputime Time to translate.

Description

Equivalent to calling: msec_to_string (cputime_to_msecs (cputime)).

Name

`function::task_time_string` — Human readable string of task time usage

Synopsis

```
task_time_string:string()
```

Arguments

None

Description

Returns a human readable string showing the user and system time the current task has used up to now. For example “usr: 0m12.908s, sys: 1m6.851s”.

Name

`function::task_time_string_tid` — Human readable string of task time usage

Synopsis

```
task_time_string_tid:string(tid:long)
```

Arguments

tid Thread id of the given task

Description

Returns a human readable string showing the user and system time the given task has used up to now.
For example “usr: 0m12.908s, sys: 1m6.851s”.

Chapter 7. IO Scheduler and block IO Tapset

This family of probe points is used to probe block IO layer and IO scheduler activities. It contains the following probe points:

Name

probe::ioscheduler.elv_next_request — Fires when a request is retrieved from the request queue

Synopsis

```
ioscheduler.elv_next_request
```

Values

<i>name</i>	Name of the probe point
<i>elevator_name</i>	The type of I/O elevator currently enabled

Name

probe::ioscheduler.elv_next_request.return — Fires when a request retrieval issues a return signal

Synopsis

```
ioscheduler.elv_next_request.return
```

Values

<i>disk_major</i>	Disk major number of the request
<i>rq</i>	Address of the request
<i>name</i>	Name of the probe point
<i>disk_minor</i>	Disk minor number of the request
<i>rq_flags</i>	Request flags

Name

probe::ioscheduler.elv_completed_request — Fires when a request is completed

Synopsis

```
ioscheduler.elv_completed_request
```

Values

<i>disk_major</i>	Disk major number of the request
<i>rq</i>	Address of the request
<i>name</i>	Name of the probe point
<i>elevator_name</i>	The type of I/O elevator currently enabled
<i>disk_minor</i>	Disk minor number of the request
<i>rq_flags</i>	Request flags

Name

probe::ioscheduler.elv_add_request.kp — kprobe based probe to indicate that a request was added to the request queue

Synopsis

```
ioscheduler.elv_add_request.kp
```

Values

<i>disk_major</i>	Disk major number of the request
<i>rq</i>	Address of the request
<i>q</i>	pointer to request queue
<i>name</i>	Name of the probe point
<i>elevator_name</i>	The type of I/O elevator currently enabled
<i>disk_minor</i>	Disk minor number of the request
<i>rq_flags</i>	Request flags

Name

probe::ioscheduler.elv_add_request.tp — tracepoint based probe to indicate a request is added to the request queue.

Synopsis

```
ioscheduler.elv_add_request.tp
```

Values

<i>disk_major</i>	Disk major no of request.
<i>rq</i>	Address of request.
<i>q</i>	Pointer to request queue.
<i>name</i>	Name of the probe point
<i>elevator_name</i>	The type of I/O elevator currently enabled.
<i>disk_minor</i>	Disk minor number of request.
<i>rq_flags</i>	Request flags.

Name

probe::ioscheduler.elv_add_request — probe to indicate request is added to the request queue.

Synopsis

```
ioscheduler.elv_add_request
```

Values

<i>disk_major</i>	Disk major no of request.
<i>rq</i>	Address of request.
<i>q</i>	Pointer to request queue.
<i>elevator_name</i>	The type of I/O elevator currently enabled.
<i>disk_minor</i>	Disk minor number of request.
<i>rq_flags</i>	Request flags.

Name

probe::ioscheduler_trace.elv_completed_request — Fires when a request is

Synopsis

```
ioscheduler_trace.elv_completed_request
```

Values

<i>disk_major</i>	Disk major no of request.
<i>rq</i>	Address of request.
<i>name</i>	Name of the probe point
<i>elevator_name</i>	The type of I/O elevator currently enabled.
<i>disk_minor</i>	Disk minor number of request.
<i>rq_flags</i>	Request flags.

Description

completed.

Name

probe::ioscheduler_trace.elv_issue_request — Fires when a request is

Synopsis

```
ioscheduler_trace.elv_issue_request
```

Values

<i>disk_major</i>	Disk major no of request.
<i>rq</i>	Address of request.
<i>name</i>	Name of the probe point
<i>elevator_name</i>	The type of I/O elevator currently enabled.
<i>disk_minor</i>	Disk minor number of request.
<i>rq_flags</i>	Request flags.

Description

scheduled.

Name

probe::ioscheduler_trace.elv_requeue_request — Fires when a request is

Synopsis

```
ioscheduler_trace.elv_requeue_request
```

Values

<i>disk_major</i>	Disk major no of request.
<i>rq</i>	Address of request.
<i>name</i>	Name of the probe point
<i>elevator_name</i>	The type of I/O elevator currently enabled.
<i>disk_minor</i>	Disk minor number of request.
<i>rq_flags</i>	Request flags.

Description

put back on the queue, when the hardware cannot accept more requests.

Name

probe::ioscheduler_trace.elv_abort_request — Fires when a request is aborted.

Synopsis

```
ioscheduler_trace.elv_abort_request
```

Values

<i>disk_major</i>	Disk major no of request.
<i>rq</i>	Address of request.
<i>name</i>	Name of the probe point
<i>elevator_name</i>	The type of I/O elevator currently enabled.
<i>disk_minor</i>	Disk minor number of request.
<i>rq_flags</i>	Request flags.

Name

probe::ioscheduler_trace.plugin — Fires when a request queue is plugged;

Synopsis

```
ioscheduler_trace.plugin
```

Values

<i>name</i>	Name of the probe point
<i>rq_queue</i>	request queue

Description

ie, requests in the queue cannot be serviced by block driver.

Name

probe::ioscheduler_trace.unplug_io — Fires when a request queue is unplugged;

Synopsis

```
ioscheduler_trace.unplug_io
```

Values

<i>name</i>	Name of the probe point
<i>rq_queue</i>	request queue

Description

Either, when number of pending requests in the queue exceeds threshold or, upon expiration of timer that was activated when queue was plugged.

Name

probe::ioscheduler_trace.unplug_timer — Fires when unplug timer associated

Synopsis

```
ioscheduler_trace.unplug_timer
```

Values

<i>name</i>	Name of the probe point
<i>rq_queue</i>	request queue

Description

with a request queue expires.

Name

probe::ioblock.request — Fires whenever making a generic block I/O request.

Synopsis

```
ioblock.request
```

Values

None

Description

name - name of the probe point *devname* - block device name *ino* - i-node number of the mapped file *sector* - beginning sector for the entire bio *flags* - see below BIO_UPTODATE 0 ok after I/O completion BIO_RW_BLOCK 1 RW_AHEAD set, and read/write would block BIO_EOF 2 out-out-bounds error BIO_SEG_VALID 3 nr_hw_seg valid BIO_CLONED 4 doesn't own data BIO_BOUNCED 5 bio is a bounce bio BIO_USER_MAPPED 6 contains user pages BIO_EOPNOTSUPP 7 not supported

rw - binary trace for read/write request *vcnt* - bio vector count which represents number of array element (page, offset, length) which make up this I/O request *idx* - offset into the bio vector array *phys_segments* - number of segments in this bio after physical address coalescing is performed *hw_segments* - number of segments after physical and DMA remapping hardware coalescing is performed *size* - total size in bytes *bdev* - target block device *bdev_contains* - points to the device object which contains the partition (when bio structure represents a partition) *p_start_sect* - points to the start sector of the partition structure of the device

Context

The process makes block I/O request

Name

probe::ioblock.end — Fires whenever a block I/O transfer is complete.

Synopsis

```
ioblock.end
```

Values

None

Description

name - name of the probe point *devname* - block device name *ino* - i-node number of the mapped file *bytes_done* - number of bytes transferred *sector* - beginning sector for the entire bio *flags* - see below BIO_UPTODATE 0 ok after I/O completion BIO_RW_BLOCK 1 RW_AHEAD set, and read/write would block BIO_EOF 2 out-out-bounds error BIO_SEG_VALID 3 nr_hw_seg valid BIO_CLONED 4 doesn't own data BIO_BOUNCED 5 bio is a bounce bio BIO_USER_MAPPED 6 contains user pages BIO_EOPNOTSUPP 7 not supported *error* - 0 on success *rw* - binary trace for read/write request *vcnt* - bio vector count which represents number of array element (page, offset, length) which makes up this I/O request *idx* - offset into the bio vector array *phys_segments* - number of segments in this bio after physical address coalescing is performed. *hw_segments* - number of segments after physical and DMA remapping hardware coalescing is performed *size* - total size in bytes

Context

The process signals the transfer is done.

Name

probe::ioblock_trace.bounce — Fires whenever a buffer bounce is needed for at least one page of a block IO request.

Synopsis

ioblock_trace.bounce

Values

None

Description

name - name of the probe point *q* - request queue on which this bio was queued. *devname* - device for which a buffer bounce was needed. *ino* - i-node number of the mapped file *bytes_done* - number of bytes transferred *sector* - beginning sector for the entire bio *flags* - see below BIO_UPTODATE 0 ok after I/O completion BIO_RW_BLOCK 1 RW_AHEAD set, and read/write would block BIO_EOF 2 out-out-bounds error BIO_SEG_VALID 3 nr_hw_seg valid BIO_CLONED 4 doesn't own data BIO_BOUNCED 5 bio is a bounce bio BIO_USER_MAPPED 6 contains user pages BIO_EOPNOTSUPP 7 not supported *rw* - binary trace for read/write request *vcnt* - bio vector count which represents number of array element (page, offset, length) which makes up this I/O request *idx* - offset into the bio vector array *phys_segments* - number of segments in this bio after physical address coalescing is performed. *size* - total size in bytes *bdev* - target block device *bdev_contains* - points to the device object which contains the partition (when bio structure represents a partition) *p_start_sect* - points to the start sector of the partition structure of the device

Context

The process creating a block IO request.

Name

probe::ioblock_trace.request — Fires just as a generic block I/O request is created for a bio.

Synopsis

```
ioblock_trace.request
```

Values

None

Description

name - name of the probe point *q* - request queue on which this bio was queued. *devname* - block device name *ino* - i-node number of the mapped file *bytes_done* - number of bytes transferred *sector* - beginning sector for the entire bio *flags* - see below BIO_UPTODATE 0 ok after I/O completion BIO_RW_BLOCK 1 RW_AHEAD set, and read/write would block BIO_EOF 2 out-out-bounds error BIO_SEG_VALID 3 nr_hw_seg valid BIO_CLONED 4 doesn't own data BIO_BOUNCED 5 bio is a bounce bio BIO_USER_MAPPED 6 contains user pages BIO_EOPNOTSUPP 7 not supported

rw - binary trace for read/write request *vcnt* - bio vector count which represents number of array element (page, offset, length) which make up this I/O request *idx* - offset into the bio vector array *phys_segments* - number of segments in this bio after physical address coalescing is performed. *size* - total size in bytes *bdev* - target block device *bdev_contains* - points to the device object which contains the partition (when bio structure represents a partition) *p_start_sect* - points to the start sector of the partition structure of the device

Context

The process makes block I/O request

Name

probe::ioblock_trace.end — Fires whenever a block I/O transfer is complete.

Synopsis

```
ioblock_trace.end
```

Values

None

Description

name - name of the probe point *q* - request queue on which this bio was queued. *devname* - block device name *ino* - i-node number of the mapped file *bytes_done* - number of bytes transferred *sector* - beginning sector for the entire bio *flags* - see below BIO_UPTODATE 0 ok after I/O completion BIO_RW_BLOCK 1 RW_AHEAD set, and read/write would block BIO_EOF 2 out-out-bounds error BIO_SEG_VALID 3 nr_hw_seg valid BIO_CLONED 4 doesn't own data BIO_BOUNCED 5 bio is a bounce bio BIO_USER_MAPPED 6 contains user pages BIO_EOPNOTSUPP 7 not supported

rw - binary trace for read/write request *vcnt* - bio vector count which represents number of array element (page, offset, length) which makes up this I/O request *idx* - offset into the bio vector array *phys_segments* - number of segments in this bio after physical address coalescing is performed. *size* - total size in bytes *bdev* - target block device *bdev_contains* - points to the device object which contains the partition (when bio structure represents a partition) *p_start_sect* - points to the start sector of the partition structure of the device

Context

The process signals the transfer is done.

Chapter 8. SCSI Tapset

This family of probe points is used to probe SCSI activities. It contains the following probe points:

Name

probe::scsi.ioentry — Prepares a SCSI mid-layer request

Synopsis

```
scsi.ioentry
```

Values

<i>disk_major</i>	The major number of the disk (-1 if no information)
<i>device_state_str</i>	The current state of the device, as a string
<i>device_state</i>	The current state of the device
<i>req_addr</i>	The current struct request pointer, as a number
<i>disk_minor</i>	The minor number of the disk (-1 if no information)

Name

probe::scsi.iodispatching — SCSI mid-layer dispatched low-level SCSI command

Synopsis

`scsi.iodispatching`

Values

<i>device_state_str</i>	The current state of the device, as a string
<i>dev_id</i>	The scsi device id
<i>channel</i>	The channel number
<i>data_direction</i>	The <code>data_direction</code> specifies whether this command is from/to the device 0 (DMA_BIDIRECTIONAL), 1 (DMA_TO_DEVICE), 2 (DMA_FROM_DEVICE), 3 (DMA_NONE)
<i>lun</i>	The lun number
<i>request_bufflen</i>	The request buffer length
<i>host_no</i>	The host number
<i>device_state</i>	The current state of the device
<i>data_direction_str</i>	Data direction, as a string
<i>req_addr</i>	The current struct request pointer, as a number
<i>request_buffer</i>	The request buffer address

Name

probe::scsi.iodone — SCSI command completed by low level driver and enqueued into the done queue.

Synopsis

```
scsi.iodone
```

Values

<i>device_state_str</i>	The current state of the device, as a string
<i>dev_id</i>	The scsi device id
<i>channel</i>	The channel number
<i>data_direction</i>	The <i>data_direction</i> specifies whether this command is from/to the device.
<i>lun</i>	The lun number
<i>host_no</i>	The host number
<i>data_direction_str</i>	Data direction, as a string
<i>device_state</i>	The current state of the device
<i>scsi_timer_pending</i>	1 if a timer is pending on this request
<i>req_addr</i>	The current struct request pointer, as a number

Name

probe::scsi.iocompleted — SCSI mid-layer running the completion processing for block device I/O requests

Synopsis

```
scsi.iocompleted
```

Values

<i>device_state_str</i>	The current state of the device, as a string
<i>dev_id</i>	The scsi device id
<i>channel</i>	The channel number
<i>data_direction</i>	The <i>data_direction</i> specifies whether this command is from/to the device
<i>lun</i>	The lun number
<i>host_no</i>	The host number
<i>data_direction_str</i>	Data direction, as a string
<i>device_state</i>	The current state of the device
<i>req_addr</i>	The current struct request pointer, as a number
<i>goodbytes</i>	The bytes completed

Name

probe::scsi.ioexecute — Create mid-layer SCSI request and wait for the result

Synopsis

```
scsi.ioexecute
```

Values

<i>retries</i>	Number of times to retry request
<i>device_state_str</i>	The current state of the device, as a string
<i>dev_id</i>	The scsi device id
<i>channel</i>	The channel number
<i>data_direction</i>	The <i>data_direction</i> specifies whether this command is from/to the device.
<i>lun</i>	The lun number
<i>timeout</i>	Request timeout in seconds
<i>request_bufflen</i>	The data buffer buffer length
<i>host_no</i>	The host number
<i>data_direction_str</i>	Data direction, as a string
<i>device_state</i>	The current state of the device
<i>request_buffer</i>	The data buffer address

Name

probe::scsi.set_state — Order SCSI device state change

Synopsis

```
scsi.set_state
```

Values

<i>state_str</i>	The new state of the device, as a string
<i>dev_id</i>	The scsi device id
<i>channel</i>	The channel number
<i>state</i>	The new state of the device
<i>old_state_str</i>	The current state of the device, as a string
<i>lun</i>	The lun number
<i>old_state</i>	The current state of the device
<i>host_no</i>	The host number

Chapter 9. TTY Tapset

This family of probe points is used to probe TTY (Teletype) activities. It contains the following probe points:

Name

probe::tty.open — Called when a tty is opened

Synopsis

```
tty.open
```

Values

<i>inode_state</i>	the inode state
<i>file_name</i>	the file name
<i>file_mode</i>	the file mode
<i>file_flags</i>	the file flags
<i>inode_number</i>	the inode number
<i>inode_flags</i>	the inode flags

Name

probe::tty.release — Called when the tty is closed

Synopsis

```
tty.release
```

Values

<i>inode_state</i>	the inode state
<i>file_name</i>	the file name
<i>file_mode</i>	the file mode
<i>file_flags</i>	the file flags
<i>inode_number</i>	the inode number
<i>inode_flags</i>	the inode flags

Name

probe::tty.resize — Called when a terminal resize happens

Synopsis

```
tty.resize
```

Values

<i>new_ypixel</i>	the new ypixel value
<i>old_col</i>	the old col value
<i>old_xpixel</i>	the old xpixel
<i>old_ypixel</i>	the old ypixel
<i>name</i>	the tty name
<i>old_row</i>	the old row value
<i>new_row</i>	the new row value
<i>new_xpixel</i>	the new xpixel value
<i>new_col</i>	the new col value

Name

probe::tty.ioctl — called when a ioctl is request to the tty

Synopsis

```
tty.ioctl
```

Values

cmd the ioctl command

arg the ioctl argument

name the file name

Name

probe::tty.init — Called when a tty is being initialized

Synopsis

```
tty.init
```

Values

<i>driver_name</i>	the driver name
<i>name</i>	the driver <code>.dev_name</code> name
<i>module</i>	the module name

Name

probe::tty.register — Called when a tty device is registred

Synopsis

```
tty.register
```

Values

<i>driver_name</i>	the driver name
<i>name</i>	the driver .dev_name name
<i>index</i>	the tty index requested
<i>module</i>	the module name

Name

probe::tty.unregister — Called when a tty device is being unregistered

Synopsis

```
tty.unregister
```

Values

<i>driver_name</i>	the driver name
<i>name</i>	the driver .dev_name name
<i>index</i>	the tty index requested
<i>module</i>	the module name

Name

probe::tty.poll — Called when a tty device is being polled

Synopsis

```
tty.poll
```

Values

<i>file_name</i>	the tty file name
<i>wait_key</i>	the wait queue key

Name

probe::tty.receive — called when a tty receives a message

Synopsis

```
tty.receive
```

Values

<i>driver_name</i>	the driver name
<i>count</i>	The amount of characters received
<i>name</i>	the name of the module file
<i>fp</i>	The flag buffer
<i>cp</i>	the buffer that was received
<i>index</i>	The tty Index
<i>id</i>	the tty id

Name

probe::tty.write — write to the tty line

Synopsis

```
tty.write
```

Values

<i>driver_name</i>	the driver name
<i>buffer</i>	the buffer that will be written
<i>file_name</i>	the file name lreated to the tty
<i>nr</i>	The amount of characters

Name

probe::tty.read — called when a tty line will be read

Synopsis

```
tty.read
```

Values

<i>driver_name</i>	the driver name
<i>buffer</i>	the buffer that will receive the characters
<i>file_name</i>	the file name lreated to the tty
<i>nr</i>	The amount of characters to be read

Chapter 10. Networking Tapset

This family of probe points is used to probe the activities of the network device and protocol layers.

Name

probe::netdev.receive — Data received from network device.

Synopsis

```
netdev.receive
```

Values

<i>protocol</i>	Protocol of received packet.
<i>dev_name</i>	The name of the device. e.g: eth0, ath1.
<i>length</i>	The length of the receiving buffer.

Name

probe::netdev.transmit — Network device transmitting buffer

Synopsis

```
netdev.transmit
```

Values

<i>protocol</i>	The protocol of this packet(defined in include/linux/if_ether.h).
<i>dev_name</i>	The name of the device. e.g: eth0, ath1.
<i>length</i>	The length of the transmit buffer.
<i>truesize</i>	The size of the data to be transmitted.

Name

probe::netdev.change_mtu — Called when the netdev MTU is changed

Synopsis

```
netdev.change_mtu
```

Values

<i>dev_name</i>	The device that will have the MTU changed
<i>new_mtu</i>	The new MTU
<i>old_mtu</i>	The current MTU

Name

probe::netdev.open — Called when the device is opened

Synopsis

```
netdev.open
```

Values

<i>dev_name</i>	The device that is going to be opened
-----------------	---------------------------------------

Name

probe::netdev.close — Called when the device is closed

Synopsis

```
netdev.close
```

Values

<i>dev_name</i>	The device that is going to be closed
-----------------	---------------------------------------

Name

probe::netdev.hard_transmit — Called when the devices is going to TX (hard)

Synopsis

```
netdev.hard_transmit
```

Values

<i>protocol</i>	The protocol used in the transmission
<i>dev_name</i>	The device scheduled to transmit
<i>length</i>	The length of the transmit buffer.
<i>truesize</i>	The size of the data to be transmitted.

Name

probe::netdev.rx — Called when the device is going to receive a packet

Synopsis

```
netdev.rx
```

Values

<i>protocol</i>	The packet protocol
<i>dev_name</i>	The device received the packet

Name

probe::netdev.change_rx_flag — Called when the device RX flag will be changed

Synopsis

```
netdev.change_rx_flag
```

Values

<i>dev_name</i>	The device that will be changed
<i>flags</i>	The new flags

Name

probe::netdev.set_promiscuity — Called when the device enters/leaves promiscuity

Synopsis

```
netdev.set_promiscuity
```

Values

<i>dev_name</i>	The device that is entering/leaving promiscuity mode
<i>enable</i>	If the device is entering promiscuity mode
<i>inc</i>	Count the number of promiscuity openers
<i>disable</i>	If the device is leaving promiscuity mode

Name

probe::netdev.ioctl — Called when the device suffers an IOCTL

Synopsis

```
netdev.ioctl
```

Values

cmd The IOCTL request

arg The IOCTL argument (usually the netdev interface)

Name

probe::netdev.register — Called when the device is registered

Synopsis

```
netdev.register
```

Values

<i>dev_name</i>	The device that is going to be registered
-----------------	---

Name

probe::netdev.unregister — Called when the device is being unregistered

Synopsis

```
netdev.unregister
```

Values

<i>dev_name</i>	The device that is going to be unregistered
-----------------	---

Name

probe::netdev.get_stats — Called when someone asks the device statistics

Synopsis

```
netdev.get_stats
```

Values

<i>dev_name</i>	The device that is going to provide the statistics
-----------------	--

Name

probe::netdev.change_mac — Called when the netdev_name has the MAC changed

Synopsis

```
netdev.change_mac
```

Values

<i>dev_name</i>	The device that will have the MTU changed
<i>new_mac</i>	The new MAC address
<i>mac_len</i>	The MAC length
<i>old_mac</i>	The current MAC address

Name

probe::tcp.sendmsg — Sending a tcp message

Synopsis

`tcp.sendmsg`

Values

<i>name</i>	Name of this probe
<i>size</i>	Number of bytes to send
<i>sock</i>	Network socket

Context

The process which sends a tcp message

Name

probe::tcp.sendmsg.return — Sending TCP message is done

Synopsis

```
tcp.sendmsg.return
```

Values

name Name of this probe

size Number of bytes sent or error code if an error occurred.

Context

The process which sends a tcp message

Name

probe::tcp.recvmsg — Receiving TCP message

Synopsis

`tcp.recvmsg`

Values

<i>saddr</i>	A string representing the source IP address
<i>daddr</i>	A string representing the destination IP address
<i>name</i>	Name of this probe
<i>sport</i>	TCP source port
<i>dport</i>	TCP destination port
<i>size</i>	Number of bytes to be received
<i>sock</i>	Network socket

Context

The process which receives a tcp message

Name

probe::tcp.recvmsg.return — Receiving TCP message complete

Synopsis

```
tcp.recvmsg.return
```

Values

<i>saddr</i>	A string representing the source IP address
<i>daddr</i>	A string representing the destination IP address
<i>name</i>	Name of this probe
<i>sport</i>	TCP source port
<i>dport</i>	TCP destination port
<i>size</i>	Number of bytes received or error code if an error occurred.

Context

The process which receives a tcp message

Name

probe::tcp.disconnect — TCP socket disconnection

Synopsis

`tcp.disconnect`

Values

<i>saddr</i>	A string representing the source IP address
<i>daddr</i>	A string representing the destination IP address
<i>flags</i>	TCP flags (e.g. FIN, etc)
<i>name</i>	Name of this probe
<i>sport</i>	TCP source port
<i>dport</i>	TCP destination port
<i>sock</i>	Network socket

Context

The process which disconnects tcp

Name

probe::tcp.disconnect.return — TCP socket disconnection complete

Synopsis

```
tcp.disconnect.return
```

Values

ret Error code (0: no error)

name Name of this probe

Context

The process which disconnects tcp

Name

probe::tcp.setsockopt — Call to setsockopt

Synopsis

`tcp.setsockopt`

Values

<i>optstr</i>	Resolves optname to a human-readable format
<i>level</i>	The level at which the socket options will be manipulated
<i>optlen</i>	Used to access values for setsockopt
<i>name</i>	Name of this probe
<i>optname</i>	TCP socket options (e.g. TCP_NODELAY, TCP_MAXSEG, etc)
<i>sock</i>	Network socket

Context

The process which calls setsockopt

Name

probe::tcp.setsockopt.return — Return from setsockopt

Synopsis

```
tcp.setsockopt.return
```

Values

ret Error code (0: no error)

name Name of this probe

Context

The process which calls setsockopt

Name

probe::tcp.receive — Called when a TCP packet is received

Synopsis

`tcp.receive`

Values

<i>urg</i>	TCP URG flag
<i>protocol</i>	Packet protocol from driver
<i>psh</i>	TCP PSH flag
<i>name</i>	Name of the probe point
<i>rst</i>	TCP RST flag
<i>dport</i>	TCP destination port
<i>saddr</i>	A string representing the source IP address
<i>daddr</i>	A string representing the destination IP address
<i>ack</i>	TCP ACK flag
<i>fin</i>	TCP FIN flag
<i>syn</i>	TCP SYN flag
<i>sport</i>	TCP source port
<i>iphdr</i>	IP header address

Name

probe::udp.sendmsg — Fires whenever a process sends a UDP message

Synopsis

```
udp.sendmsg
```

Values

<i>name</i>	The name of this probe
<i>size</i>	Number of bytes sent by the process
<i>sock</i>	Network socket used by the process

Context

The process which sent a UDP message

Name

probe::udp.sendmsg.return — Fires whenever an attempt to send a UDP message is completed

Synopsis

```
udp.sendmsg.return
```

Values

<i>name</i>	The name of this probe
<i>size</i>	Number of bytes sent by the process

Context

The process which sent a UDP message

Name

probe::udp.recvmsg — Fires whenever a UDP message is received

Synopsis

```
udp.recvmsg
```

Values

<i>name</i>	The name of this probe
<i>size</i>	Number of bytes received by the process
<i>sock</i>	Network socket used by the process

Context

The process which received a UDP message

Name

probe::udp.recvmsg.return — Fires whenever an attempt to receive a UDP message received is completed

Synopsis

```
udp.recvmsg.return
```

Values

name The name of this probe

size Number of bytes received by the process

Context

The process which received a UDP message

Name

probe::udp.disconnect — Fires when a process requests for a UDP disconnection

Synopsis

```
udp.disconnect
```

Values

<i>flags</i>	Flags (e.g. FIN, etc)
<i>name</i>	The name of this probe
<i>sock</i>	Network socket used by the process

Context

The process which requests a UDP disconnection

Name

probe::udp.disconnect.return — UDP has been disconnected successfully

Synopsis

```
udp.disconnect.return
```

Values

ret Error code (0: no error)

name The name of this probe

Context

The process which requested a UDP disconnection

Name

function::ip_ntop — returns a string representation from an integer IP number

Synopsis

```
ip_ntop:string(addr:long)
```

Arguments

addr the ip represented as an integer

Chapter 11. Socket Tapset

This family of probe points is used to probe socket activities. It contains the following probe points:

Name

probe::socket.send — Message sent on a socket.

Synopsis

```
socket.send
```

Values

<i>success</i>	Was send successful? (1 = yes, 0 = no)
<i>protocol</i>	Protocol value
<i>flags</i>	Socket flags value
<i>name</i>	Name of this probe
<i>state</i>	Socket state value
<i>size</i>	Size of message sent (in bytes) or error code if success = 0
<i>type</i>	Socket type value
<i>family</i>	Protocol family value

Context

The message sender

Name

probe::socket.receive — Message received on a socket.

Synopsis

```
socket.receive
```

Values

<i>success</i>	Was send successful? (1 = yes, 0 = no)
<i>protocol</i>	Protocol value
<i>flags</i>	Socket flags value
<i>name</i>	Name of this probe
<i>state</i>	Socket state value
<i>size</i>	Size of message received (in bytes) or error code if success = 0
<i>type</i>	Socket type value
<i>family</i>	Protocol family value

Context

The message receiver

Name

probe::socket.sendmsg — Message is currently being sent on a socket.

Synopsis

```
socket.sendmsg
```

Values

<i>protocol</i>	Protocol value
<i>flags</i>	Socket flags value
<i>name</i>	Name of this probe
<i>state</i>	Socket state value
<i>size</i>	Message size in bytes
<i>type</i>	Socket type value
<i>family</i>	Protocol family value

Context

The message sender

Description

Fires at the beginning of sending a message on a socket via the `sock_sendmsg` function

Name

probe::socket.sendmsg.return — Return from socket.sendmsg.

Synopsis

```
socket.sendmsg.return
```

Values

<i>success</i>	Was send successful? (1 = yes, 0 = no)
<i>protocol</i>	Protocol value
<i>flags</i>	Socket flags value
<i>name</i>	Name of this probe
<i>state</i>	Socket state value
<i>size</i>	Size of message sent (in bytes) or error code if success = 0
<i>type</i>	Socket type value
<i>family</i>	Protocol family value

Context

The message sender.

Description

Fires at the conclusion of sending a message on a socket via the `sock_sendmsg` function

Name

probe::socket.recvmsg — Message being received on socket

Synopsis

```
socket.recvmsg
```

Values

<i>protocol</i>	Protocol value
<i>flags</i>	Socket flags value
<i>name</i>	Name of this probe
<i>state</i>	Socket state value
<i>size</i>	Message size in bytes
<i>type</i>	Socket type value
<i>family</i>	Protocol family value

Context

The message receiver.

Description

Fires at the beginning of receiving a message on a socket via the `sock_recvmsg` function

Name

probe::socket.recvmsg.return — Return from Message being received on socket

Synopsis

```
socket.recvmsg.return
```

Values

<i>success</i>	Was receive successful? (1 = yes, 0 = no)
<i>protocol</i>	Protocol value
<i>flags</i>	Socket flags value
<i>name</i>	Name of this probe
<i>state</i>	Socket state value
<i>size</i>	Size of message received (in bytes) or error code if success = 0
<i>type</i>	Socket type value
<i>family</i>	Protocol family value

Context

The message receiver.

Description

Fires at the conclusion of receiving a message on a socket via the `sock_recvmsg` function.

Name

probe::socket.aio_write — Message send via `sock_aio_write`

Synopsis

```
socket.aio_write
```

Values

<i>protocol</i>	Protocol value
<i>flags</i>	Socket flags value
<i>name</i>	Name of this probe
<i>state</i>	Socket state value
<i>size</i>	Message size in bytes
<i>type</i>	Socket type value
<i>family</i>	Protocol family value

Context

The message sender

Description

Fires at the beginning of sending a message on a socket via the `sock_aio_write` function

Name

probe::socket.aio_write.return — Conclusion of message send via `sock_aio_write`

Synopsis

```
socket.aio_write.return
```

Values

<i>success</i>	Was receive successful? (1 = yes, 0 = no)
<i>protocol</i>	Protocol value
<i>flags</i>	Socket flags value
<i>name</i>	Name of this probe
<i>state</i>	Socket state value
<i>size</i>	Size of message received (in bytes) or error code if success = 0
<i>type</i>	Socket type value
<i>family</i>	Protocol family value

Context

The message receiver.

Description

Fires at the conclusion of sending a message on a socket via the `sock_aio_write` function

Name

probe::socket.aio_read — Receiving message via `sock_aio_read`

Synopsis

```
socket.aio_read
```

Values

<i>protocol</i>	Protocol value
<i>flags</i>	Socket flags value
<i>name</i>	Name of this probe
<i>state</i>	Socket state value
<i>size</i>	Message size in bytes
<i>type</i>	Socket type value
<i>family</i>	Protocol family value

Context

The message sender

Description

Fires at the beginning of receiving a message on a socket via the `sock_aio_read` function

Name

probe::socket.aio_read.return — Conclusion of message received via `sock_aio_read`

Synopsis

```
socket.aio_read.return
```

Values

<i>success</i>	Was receive successful? (1 = yes, 0 = no)
<i>protocol</i>	Protocol value
<i>flags</i>	Socket flags value
<i>name</i>	Name of this probe
<i>state</i>	Socket state value
<i>size</i>	Size of message received (in bytes) or error code if success = 0
<i>type</i>	Socket type value
<i>family</i>	Protocol family value

Context

The message receiver.

Description

Fires at the conclusion of receiving a message on a socket via the `sock_aio_read` function

Name

probe::socket.writev — Message sent via `socket_writev`

Synopsis

```
socket.writev
```

Values

<i>protocol</i>	Protocol value
<i>flags</i>	Socket flags value
<i>name</i>	Name of this probe
<i>state</i>	Socket state value
<i>size</i>	Message size in bytes
<i>type</i>	Socket type value
<i>family</i>	Protocol family value

Context

The message sender

Description

Fires at the beginning of sending a message on a socket via the `sock_writev` function

Name

`probe::socket.writev.return` — Conclusion of message sent via `socket_writev`

Synopsis

`socket.writev.return`

Values

<i>success</i>	Was send successful? (1 = yes, 0 = no)
<i>protocol</i>	Protocol value
<i>flags</i>	Socket flags value
<i>name</i>	Name of this probe
<i>state</i>	Socket state value
<i>size</i>	Size of message sent (in bytes) or error code if success = 0
<i>type</i>	Socket type value
<i>family</i>	Protocol family value

Context

The message receiver.

Description

Fires at the conclusion of sending a message on a socket via the `sock_writev` function

Name

probe::socket.readv — Receiving a message via `sock_readv`

Synopsis

`socket.readv`

Values

<i>protocol</i>	Protocol value
<i>flags</i>	Socket flags value
<i>name</i>	Name of this probe
<i>state</i>	Socket state value
<i>size</i>	Message size in bytes
<i>type</i>	Socket type value
<i>family</i>	Protocol family value

Context

The message sender

Description

Fires at the beginning of receiving a message on a socket via the `sock_readv` function

Name

probe::socket.readv.return — Conclusion of receiving a message via `sock_readv`

Synopsis

```
socket.readv.return
```

Values

<i>success</i>	Was receive successful? (1 = yes, 0 = no)
<i>protocol</i>	Protocol value
<i>flags</i>	Socket flags value
<i>name</i>	Name of this probe
<i>state</i>	Socket state value
<i>size</i>	Size of message received (in bytes) or error code if success = 0
<i>type</i>	Socket type value
<i>family</i>	Protocol family value

Context

The message receiver.

Description

Fires at the conclusion of receiving a message on a socket via the `sock_readv` function

Name

probe::socket.create — Creation of a socket

Synopsis

```
socket.create
```

Values

<i>protocol</i>	Protocol value
<i>name</i>	Name of this probe
<i>requester</i>	Requested by user process or the kernel (1 = kernel, 0 = user)
<i>type</i>	Socket type value
<i>family</i>	Protocol family value

Context

The requester (see requester variable)

Description

Fires at the beginning of creating a socket.

Name

probe::socket.create.return — Return from Creation of a socket

Synopsis

```
socket.create.return
```

Values

<i>success</i>	Was socket creation successful? (1 = yes, 0 = no)
<i>protocol</i>	Protocol value
<i>err</i>	Error code if success == 0
<i>name</i>	Name of this probe
<i>requester</i>	Requested by user process or the kernel (1 = kernel, 0 = user)
<i>type</i>	Socket type value
<i>family</i>	Protocol family value

Context

The requester (user process or kernel)

Description

Fires at the conclusion of creating a socket.

Name

probe::socket.close — Close a socket

Synopsis

```
socket.close
```

Values

<i>protocol</i>	Protocol value
<i>flags</i>	Socket flags value
<i>name</i>	Name of this probe
<i>state</i>	Socket state value
<i>type</i>	Socket type value
<i>family</i>	Protocol family value

Context

The requester (user process or kernel)

Description

Fires at the beginning of closing a socket.

Name

probe::socket.close.return — Return from closing a socket

Synopsis

```
socket.close.return
```

Values

name Name of this probe

Context

The requester (user process or kernel)

Description

Fires at the conclusion of closing a socket.

Name

`function::sock_prot_num2str` — Given a protocol number, return a string representation.

Synopsis

```
sock_prot_num2str:string(proto:long)
```

Arguments

proto The protocol number.

Name

function::sock_prot_str2num — Given a protocol name (string), return the corresponding protocol number.

Synopsis

```
sock_prot_str2num:long(proto:string)
```

Arguments

proto The protocol name.

Name

`function::sock_fam_num2str` — Given a protocol family number, return a string representation.

Synopsis

```
sock_fam_num2str:string(family:long)
```

Arguments

family The family number.

Name

function::sock_fam_str2num — Given a protocol family name (string), return the corresponding

Synopsis

```
sock_fam_str2num:long(family:string)
```

Arguments

family The family name.

Description

protocol family number.

Name

`function::sock_state_num2str` — Given a socket state number, return a string representation.

Synopsis

```
sock_state_num2str:string(state:long)
```

Arguments

state The state number.

Name

`function::sock_state_str2num` — Given a socket state string, return the corresponding state number.

Synopsis

```
sock_state_str2num:long(state:string)
```

Arguments

state The state name.

Chapter 12. Kernel Process Tapset

This family of probe points is used to probe process-related activities. It contains the following probe points:

Name

probe::kprocess.create — Fires whenever a new process is successfully created

Synopsis

```
kprocess.create
```

Values

new_pid The PID of the newly created process

Context

Parent of the created process.

Description

Fires whenever a new process is successfully created, either as a result of fork (or one of its syscall variants), or a new kernel thread.

Name

probe::kprocess.start — Starting new process

Synopsis

```
kprocess.start
```

Values

None

Context

Newly created process.

Description

Fires immediately before a new process begins execution.

Name

probe::kprocess.exec — Attempt to exec to a new program

Synopsis

```
kprocess.exec
```

Values

<i>filename</i>	The path to the new executable
-----------------	--------------------------------

Context

The caller of exec.

Description

Fires whenever a process attempts to exec to a new program.

Name

probe::kprocess.exec_complete — Return from exec to a new program

Synopsis

```
kprocess.exec_complete
```

Values

<i>success</i>	A boolean indicating whether the exec was successful
<i>errno</i>	The error number resulting from the exec

Context

On success, the context of the new executable. On failure, remains in the context of the caller.

Description

Fires at the completion of an exec call.

Name

probe::kprocess.exit — Exit from process

Synopsis

```
kprocess.exit
```

Values

code The exit code of the process

Context

The process which is terminating.

Description

Fires when a process terminates. This will always be followed by a `kprocess.release`, though the latter may be delayed if the process waits in a zombie state.

Name

probe::kprocess.release — Process released

Synopsis

```
kprocess.release
```

Values

pid PID of the process being released

task A task handle to the process being released

Context

The context of the parent, if it wanted notification of this process' termination, else the context of the process itself.

Description

Fires when a process is released from the kernel. This always follows a `kprocess.exit`, though it may be delayed somewhat if the process waits in a zombie state.

Chapter 13. Signal Tapset

This family of probe points is used to probe signal activities. It contains the following probe points:

Name

probe::signal.send — Signal being sent to a process

Synopsis

```
signal.send
```

Values

<i>send2queue</i>	Indicates whether the signal is sent to an existing sigqueue
<i>name</i>	The name of the function used to send out the signal
<i>task</i>	A task handle to the signal recipient
<i>sinfo</i>	The address of siginfo struct
<i>si_code</i>	Indicates the signal type
<i>sig_name</i>	A string representation of the signal
<i>sig</i>	The number of the signal
<i>shared</i>	Indicates whether the signal is shared by the thread group
<i>sig_pid</i>	The PID of the process receiving the signal
<i>pid_name</i>	The name of the signal recipient

Context

The signal's sender.

Name

probe::signal.send.return — Signal being sent to a process completed

Synopsis

```
signal.send.return
```

Values

<i>retstr</i>	The return value to either <code>__group_send_sig_info</code> , <code>specific_send_sig_info</code> , or <code>send_sigqueue</code>
<i>send2queue</i>	Indicates whether the sent signal was sent to an existing sigqueue
<i>name</i>	The name of the function used to send out the signal
<i>shared</i>	Indicates whether the sent signal is shared by the thread group.

Context

The signal's sender. (correct?)

Description

Possible `__group_send_sig_info` and `specific_send_sig_info` return values are as follows;

0 -- The signal is successfully sent to a process,

which means that

(1) the signal was ignored by the receiving process, (2) this is a non-RT signal and the system already has one queued, and (3) the signal was successfully added to the sigqueue of the receiving process.

-EAGAIN -- The sigqueue of the receiving process is overflowing, the signal was RT, and the signal was sent by a user using something other than `kill`.

Possible `send_group_sigqueue` and `send_sigqueue` return values are as follows;

0 -- The signal was either successfully added into the sigqueue of the receiving process, or a `SI_TIMER` entry is already queued (in which case, the overrun count will be simply incremented).

1 -- The signal was ignored by the receiving process.

-1 -- (`send_sigqueue` only) The task was marked exiting, allowing `* posix_timer_event` to redirect it to the group leader.

Name

probe::signal.checkperm — Check being performed on a sent signal

Synopsis

```
signal.checkperm
```

Values

<i>name</i>	Name of the probe point
<i>task</i>	A task handle to the signal recipient
<i>sinfo</i>	The address of the siginfo structure
<i>si_code</i>	Indicates the signal type
<i>sig_name</i>	A string representation of the signal
<i>sig</i>	The number of the signal
<i>pid_name</i>	Name of the process receiving the signal
<i>sig_pid</i>	The PID of the process receiving the signal

Name

probe::signal.checkperm.return — Check performed on a sent signal completed

Synopsis

```
signal.checkperm.return
```

Values

<i>retstr</i>	Return value as a string
<i>name</i>	Name of the probe point

Name

probe::signal.wakeup — Sleeping process being wakened for signal

Synopsis

```
signal.wakeup
```

Values

<i>resume</i>	Indicates whether to wake up a task in a STOPPED or TRACED state
<i>state_mask</i>	A string representation indicating the mask of task states to wake. Possible values are TASK_INTERRUPTIBLE, TASK_STOPPED, TASK_TRACED, and TASK_INTERRUPTIBLE.
<i>pid_name</i>	Name of the process to wake
<i>sig_pid</i>	The PID of the process to wake

Name

probe::signal.check_ignored — Checking to see signal is ignored

Synopsis

```
signal.check_ignored
```

Values

<i>sig_name</i>	A string representation of the signal
<i>sig</i>	The number of the signal
<i>pid_name</i>	Name of the process receiving the signal
<i>sig_pid</i>	The PID of the process receiving the signal

Name

probe::signal.check_ignored.return — Check to see signal is ignored completed

Synopsis

```
signal.check_ignored.return
```

Values

<i>retstr</i>	Return value as a string
<i>name</i>	Name of the probe point

Name

probe::signal.force_segv — Forcing send of SIGSEGV

Synopsis

```
signal.force_segv
```

Values

<i>name</i>	Name of the probe point
<i>sig_name</i>	A string representation of the signal
<i>sig</i>	The number of the signal
<i>pid_name</i>	Name of the process receiving the signal
<i>sig_pid</i>	The PID of the process receiving the signal

Name

probe::signal.force_segv.return — Forcing send of SIGSEGV complete

Synopsis

```
signal.force_segv.return
```

Values

<i>retstr</i>	Return value as a string
<i>name</i>	Name of the probe point

Name

probe::signal.syskill — Sending kill signal to a process

Synopsis

```
signal.syskill
```

Values

<i>name</i>	Name of the probe point
<i>sig_name</i>	A string representation of the signal
<i>sig</i>	The specific signal sent to the process
<i>pid_name</i>	The name of the signal recipient
<i>sig_pid</i>	The PID of the process receiving the signal

Name

probe::signal.syskill.return — Sending kill signal completed

Synopsis

```
signal.syskill.return
```

Values

None

Name

probe::signal.sys_tkill — Sending a kill signal to a thread

Synopsis

```
signal.sys_tkill
```

Values

<i>name</i>	Name of the probe point
<i>sig_name</i>	A string representation of the signal
<i>sig</i>	The specific signal sent to the process
<i>pid_name</i>	The name of the signal recipient
<i>sig_pid</i>	The PID of the process receiving the kill signal

Description

The tkill call is analogous to kill(2), except that it also allows a process within a specific thread group to be targeted. Such processes are targeted through their unique thread IDs (TID).

Name

probe::signal.systkill.return — Sending kill signal to a thread completed

Synopsis

```
signal.systkill.return
```

Values

<i>retstr</i>	The return value to either <code>__group_send_sig_info</code> ,
<i>name</i>	Name of the probe point

Name

probe::signal.sys_tgkill — Sending kill signal to a thread group

Synopsis

```
signal.sys_tgkill
```

Values

<i>name</i>	Name of the probe point
<i>sig_name</i>	A string representation of the signal
<i>sig</i>	The specific kill signal sent to the process
<i>tgid</i>	The thread group ID of the thread receiving the kill signal
<i>pid_name</i>	The name of the signal recipient
<i>sig_pid</i>	The PID of the thread receiving the kill signal

Description

The tgkill call is similar to tkill, except that it also allows the caller to specify the thread group ID of the thread to be signalled. This protects against TID reuse.

Name

probe::signal.sys_tgkill.return — Sending kill signal to a thread group completed

Synopsis

```
signal.sys_tgkill.return
```

Values

<i>retstr</i>	The return value to either <code>__group_send_sig_info</code> ,
<i>name</i>	Name of the probe point

Name

probe::signal.send_sig_queue — Queuing a signal to a process

Synopsis

```
signal.send_sig_queue
```

Values

<i>sigqueue_addr</i>	The address of the signal queue
<i>name</i>	Name of the probe point
<i>sig_name</i>	A string representation of the signal
<i>sig</i>	The queued signal
<i>pid_name</i>	Name of the process to which the signal is queued
<i>sig_pid</i>	The PID of the process to which the signal is queued

Name

probe::signal.send_sig_queue.return — Queuing a signal to a process completed

Synopsis

```
signal.send_sig_queue.return
```

Values

<i>retstr</i>	Return value as a string
<i>name</i>	Name of the probe point

Name

probe::signal.pending — Examining pending signal

Synopsis

```
signal.pending
```

Values

<i>name</i>	Name of the probe point
<i>sigset_size</i>	The size of the user-space signal set
<i>sigset_add</i>	The address of the user-space signal set (sigset_t)

Description

This probe is used to examine a set of signals pending for delivery to a specific thread. This normally occurs when the `do_sigpending` kernel function is executed.

Name

probe::signal.pending.return — Examination of pending signal completed

Synopsis

```
signal.pending.return
```

Values

<i>retstr</i>	Return value as a string
<i>name</i>	Name of the probe point

Name

probe::signal.handle — Signal handler being invoked

Synopsis

```
signal.handle
```

Values

<i>regs</i>	The address of the kernel-mode stack area
<i>sig_code</i>	The <code>si_code</code> value of the <code>siginfo</code> signal
<i>name</i>	Name of the probe point
<i>sig_mode</i>	Indicates whether the signal was a user-mode or kernel-mode signal
<i>sinfo</i>	The address of the <code>siginfo</code> table
<i>sig_name</i>	A string representation of the signal
<i>oldset_addr</i>	The address of the bitmask array of blocked signals
<i>sig</i>	The signal number that invoked the signal handler
<i>ka_addr</i>	The address of the <code>k_sigaction</code> table associated with the signal

Name

probe::signal.handle.return — Signal handler invocation completed

Synopsis

```
signal.handle.return
```

Values

<i>retstr</i>	Return value as a string
<i>name</i>	Name of the probe point

Name

probe::signal.do_action — Examining or changing a signal action

Synopsis

```
signal.do_action
```

Values

<i>sa_mask</i>	The new mask of the signal
<i>name</i>	Name of the probe point
<i>sig_name</i>	A string representation of the signal
<i>oldsigact_addr</i>	The address of the old sigaction struct associated with the signal
<i>sig</i>	The signal to be examined/changed
<i>sa_handler</i>	The new handler of the signal
<i>sigact_addr</i>	The address of the new sigaction struct associated with the signal

Name

probe::signal.do_action.return — Examining or changing a signal action completed

Synopsis

```
signal.do_action.return
```

Values

<i>retstr</i>	Return value as a string
<i>name</i>	Name of the probe point

Name

probe::signal.procmask — Examining or changing blocked signals

Synopsis

```
signal.procmask
```

Values

<i>how</i>	Indicates how to change the blocked signals; possible values are SIG_BLOCK=0 (for blocking signals), SIG_UNBLOCK=1 (for unblocking signals), and SIG_SETMASK=2 for setting the signal mask.
<i>name</i>	Name of the probe point
<i>oldsigset_addr</i>	The old address of the signal set (sigset_t)
<i>sigset</i>	The actual value to be set for sigset_t (correct?)
<i>sigset_addr</i>	The address of the signal set (sigset_t) to be implemented

Name

probe::signal.procmask.return — Examining or changing blocked signals completed

Synopsis

```
signal.procmask.return
```

Values

<i>retstr</i>	Return value as a string
<i>name</i>	Name of the probe point

Name

probe::signal.flush — Flushing all pending signals for a task

Synopsis

```
signal.flush
```

Values

<i>name</i>	Name of the probe point
<i>task</i>	The task handler of the process performing the flush
<i>pid_name</i>	The name of the process associated with the task performing the flush
<i>sig_pid</i>	The PID of the process associated with the task performing the flush

Chapter 14. Directory-entry (dentry) Tapset

This family of functions is used to map kernel VFS directory entry pointers to file or full path names.

Name

function::d_name — get the dirent name

Synopsis

```
d_name:string(dentry:long)
```

Arguments

dentry Pointer to dentry.

Description

Returns the dirent name (path basename).

Name

function::reverse_path_walk — get the full dirent path

Synopsis

```
reverse_path_walk:string(dentry:long)
```

Arguments

dentry Pointer to dentry.

Description

Returns the path name (partial path to mount point).

Name

function::d_path — get the full nameidata path

Synopsis

```
d_path:string(nd:long)
```

Arguments

nd Pointer to nameidata.

Description

Returns the full dirent name (full path to the root), like the kernel d_path function.

Chapter 15. Logging Tapset

This family of functions is used to send simple message strings to various destinations.

Name

function::log — Send a line to the common trace buffer.

Synopsis

```
log(msg:string)
```

Arguments

msg The formatted message string.

General Syntax

```
log(msg:string)
```

Description

This function logs data. `log` sends the message immediately to `staprun` and to the bulk transport (relays) if it is being used. If the last character given is not a newline, then one is added. This function is not as efficient as `printf` and should be used only for urgent messages.

Name

`function::warn` — Send a line to the warning stream.

Synopsis

```
warn(msg:string)
```

Arguments

msg The formatted message string.

General Syntax

```
warn(msg:string)
```

Description

This function sends a warning message immediately to `staprun`. It is also sent over the bulk transport (`relayfs`) if it is being used. If the last character is not a newline, the one is added.

Name

function::exit — Start shutting down probing script.

Synopsis

```
exit()
```

Arguments

None

General Syntax

```
exit
```

Description

This only enqueues a request to start shutting down the script. New probes will not fire (except “end” probes), but all currently running ones may complete their work.

Name

`function::error` — Send an error message.

Synopsis

```
error(msg:string)
```

Arguments

msg The formatted message string.

Description

An implicit end-of-line is added. `staprun` prepends the string “ERROR:”. Sending an error message aborts the currently running probe. Depending on the `MAXERRORS` parameter, it may trigger an `exit`.

Name

function::ftrace — Send a message to the ftrace ring-buffer.

Synopsis

```
ftrace(msg:string)
```

Arguments

msg The formatted message string.

Description

If the ftrace ring-buffer is configured & available, see `/debugfs/tracing/trace` for the message. Otherwise, the message may be quietly dropped. An implicit end-of-line is added.

Chapter 16. Random functions Tapset

These functions deal with random number generation.

Name

`function::randint` — Return a random number between [0,n)

Synopsis

```
randint:long (n:long)
```

Arguments

n Number past upper limit of range, not larger than 2**20.

Chapter 17. String and data retrieving functions Tapset

Functions to retrieve strings and other primitive types from the kernel or a user space programs based on addresses. All strings are of a maximum length given by MAXSTRINGLEN.

Name

`function::kernel_string` — Retrieves string from kernel memory.

Synopsis

```
kernel_string:string(addr:long)
```

Arguments

addr The kernel address to retrieve the string from.

General Syntax

```
kernel_string:string(addr:long)
```

Description

This function returns the null terminated C string from a given kernel memory address. Reports an error on string copy fault.

Name

function::kernel_string2 — Retrieves string from kernel memory with alternative error string.

Synopsis

```
kernel_string2:string(addr:long, err_msg:string)
```

Arguments

addr The kernel address to retrieve the string from.

err_msg The error message to return when data isn't available.

General Syntax

```
kernel_string2:string(addr:long, err_msg:string)
```

Description

This function returns the null terminated C string from a given kernel memory address. Reports the given error message on string copy fault.

Name

function::kernel_string_n — Retrieves string of given length from kernel memory.

Synopsis

```
kernel_string_n:string(addr:long, n:long)
```

Arguments

addr The kernel address to retrieve the string from.

n The maximum length of the string (if not null terminated).

General Syntax

```
kernel_string_n:string(addr:long, n:long)
```

Description

Returns the C string of a maximum given length from a given kernel memory address. Reports an error on string copy fault.

Name

`function::kernel_long` — Retrieves a long value stored in kernel memory.

Synopsis

```
kernel_long:long(addr:long)
```

Arguments

addr The kernel address to retrieve the long from.

General Syntax

```
kernel_long:long(addr:long)
```

Description

Returns the long value from a given kernel memory address. Reports an error when reading from the given address fails.

Name

`function::kernel_int` — Retrieves an int value stored in kernel memory.

Synopsis

```
kernel_int:long(addr:long)
```

Arguments

addr The kernel address to retrieve the int from.

Description

Returns the int value from a given kernel memory address. Reports an error when reading from the given address fails.

Name

`function::kernel_short` — Retrieves a short value stored in kernel memory.

Synopsis

```
kernel_short:long(addr:long)
```

Arguments

addr The kernel address to retrieve the short from.

General Syntax

```
kernel_short:long(addr:long)
```

Description

Returns the short value from a given kernel memory address. Reports an error when reading from the given address fails.

Name

`function::kernel_char` — Retrieves a char value stored in kernel memory.

Synopsis

```
kernel_char:long(addr:long)
```

Arguments

addr The kernel address to retrieve the char from.

General Syntax

```
kernel_char:long(addr:long)
```

Description

Returns the char value from a given kernel memory address. Reports an error when reading from the given address fails.

Name

`function::kernel_pointer` — Retrieves a pointer value stored in kernel memory.

Synopsis

```
kernel_pointer:long(addr:long)
```

Arguments

addr The kernel address to retrieve the pointer from.

General Syntax

```
kernel_pointer:long(addr:long)
```

Description

Returns the pointer value from a given kernel memory address. Reports an error when reading from the given address fails.

Name

`function::user_string` — Retrieves string from user space.

Synopsis

```
user_string:string(addr:long)
```

Arguments

addr The user space address to retrieve the string from.

General Syntax

```
user_string:string(addr:long)
```

Description

Returns the null terminated C string from a given user space memory address. Reports “<unknown>” on the rare cases when userspace data is not accessible.

Name

`function::user_string2` — Retrieves string from user space with alternative error string.

Synopsis

```
user_string2:string(addr:long, err_msg:string)
```

Arguments

<i>addr</i>	The user space address to retrieve the string from.
<i>err_msg</i>	The error message to return when data isn't available.

General Syntax

```
user_string2:string(addr:long, err_msg:string)
```

Description

Returns the null terminated C string from a given user space memory address. Reports the given error message on the rare cases when userspace data is not accessible.

Name

`function::user_string_warn` — Retrieves string from user space.

Synopsis

```
user_string_warn:string(addr:long)
```

Arguments

addr The user space address to retrieve the string from.

General Syntax

```
user_string_warn:string(addr:long)
```

Description

Returns the null terminated C string from a given user space memory address. Reports “<unknown>” on the rare cases when userspace data is not accessible and warns (but does not abort) about the failure.

Name

function::user_string_quoted — Retrieves and quotes string from user space.

Synopsis

```
user_string_quoted:string(addr:long)
```

Arguments

addr The user space address to retrieve the string from.

General Syntax

```
user_string_quoted:string(addr:long)
```

Description

Returns the null terminated C string from a given user space memory address where any ASCII characters that are not printable are replaced by the corresponding escape sequence in the returned string. Reports “NULL” for address zero. Returns “<unknown>” on the rare cases when userspace data is not accessible at the given address.

Name

`function::user_string_n` — Retrieves string of given length from user space.

Synopsis

```
user_string_n:string(addr:long, n:long)
```

Arguments

addr The user space address to retrieve the string from.

n The maximum length of the string (if not null terminated).

General Syntax

```
user_string_n:string(addr:long, n:long)
```

Description

Returns the C string of a maximum given length from a given user space address. Returns “<unknown>” on the rare cases when userspace data is not accessible at the given address.

Name

function::user_string_n2 — Retrieves string of given length from user space.

Synopsis

```
user_string_n2:string(addr:long,n:long,err_msg:string)
```

Arguments

<i>addr</i>	The user space address to retrieve the string from.
<i>n</i>	The maximum length of the string (if not null terminated).
<i>err_msg</i>	The error message to return when data isn't available.

General Syntax

```
user_string_n2:string(addr:long, n:long, err_msg:string)
```

Description

Returns the C string of a maximum given length from a given user space address. Returns the given error message string on the rare cases when userspace data is not accessible at the given address.

Name

`function::user_string_n_warn` — Retrieves string from user space.

Synopsis

```
user_string_n_warn:string(addr:long,n:long)
```

Arguments

addr The user space address to retrieve the string from.

n The maximum length of the string (if not null terminated).

General Syntax

```
user_string_n_warn:string(addr:long, n:long)
```

Description

Returns up to *n* characters of a C string from a given user space memory address. Reports “<unknown>” on the rare cases when userspace data is not accessible and warns (but does not abort) about the failure.

Name

`function::user_string_n_quoted` — Retrieves and quotes string from user space.

Synopsis

```
user_string_n_quoted:string(addr:long, n:long)
```

Arguments

addr The user space address to retrieve the string from.

n The maximum length of the string (if not null terminated).

General Syntax

```
user_string_n_quoted:string(addr:long, n:long)
```

Description

Returns up to *n* characters of a C string from the given user space memory address where any ASCII characters that are not printable are replaced by the corresponding escape sequence in the returned string. Reports “NULL” for address zero. Returns “<unknown>” on the rare cases when userspace data is not accessible at the given address.

Name

`function::user_short` — Retrieves a short value stored in user space.

Synopsis

```
user_short:long(addr:long)
```

Arguments

addr The user space address to retrieve the short from.

General Syntax

```
user_short:long(addr:long)
```

Description

Returns the short value from a given user space address. Returns zero when user space data is not accessible.

Name

`function::user_short_warn` — Retrieves a short value stored in user space.

Synopsis

```
user_short_warn:long(addr:long)
```

Arguments

addr The user space address to retrieve the short from.

General Syntax

```
user_short_warn:long(addr:long)
```

Description

Returns the short value from a given user space address. Returns zero when user space and warns (but does not abort) about the failure.

Name

`function::user_int` — Retrieves an int value stored in user space.

Synopsis

```
user_int:long(addr:long)
```

Arguments

addr The user space address to retrieve the int from.

General Syntax

```
user_int:long(addr:long)
```

Description

Returns the int value from a given user space address. Returns zero when user space data is not accessible.

Name

`function::user_int_warn` — Retrieves an int value stored in user space.

Synopsis

```
user_int_warn:long(addr:long)
```

Arguments

addr The user space address to retrieve the int from.

General Syntax

```
user_ing_warn:long(addr:long)
```

Description

Returns the int value from a given user space address. Returns zero when user space and warns (but does not abort) about the failure.

Name

`function::user_long` — Retrieves a long value stored in user space.

Synopsis

```
user_long:long(addr:long)
```

Arguments

addr The user space address to retrieve the long from.

General Syntax

```
user_long:long(addr:long)
```

Description

Returns the long value from a given user space address. Returns zero when user space data is not accessible. Note that the size of the long depends on the architecture of the current user space task (for those architectures that support both 64/32 bit compat tasks).

Name

`function::user_long_warn` — Retrieves a long value stored in user space.

Synopsis

```
user_long_warn:long(addr:long)
```

Arguments

addr The user space address to retrieve the long from.

General Syntax

```
user_long_warn:long(addr:long)
```

Description

Returns the long value from a given user space address. Returns zero when user space and warns (but does not abort) about the failure. Note that the size of the long depends on the architecture of the current user space task (for those architectures that support both 64/32 bit compat tasks).

Name

`function::user_char` — Retrieves a char value stored in user space.

Synopsis

```
user_char:long(addr:long)
```

Arguments

addr The user space address to retrieve the char from.

General Syntax

```
user_char:long(addr:long)
```

Description

Returns the char value from a given user space address. Returns zero when user space data is not accessible.

Name

`function::user_char_warn` — Retrieves a char value stored in user space.

Synopsis

```
user_char_warn:long(addr:long)
```

Arguments

addr The user space address to retrieve the char from.

General Syntax

```
user_char_warn:long(addr:long)
```

Description

Returns the char value from a given user space address. Returns zero when user space and warns (but does not abort) about the failure.

Chapter 18. A collection of standard string functions

Functions to get the length, a substring, getting at individual characters, string searching, escaping, tokenizing, and converting strings to longs.

Name

`function::strlen` — Returns the length of a string.

Synopsis

```
strlen:long(s:string)
```

Arguments

s the string

General Syntax

```
strlen: long (str:string)
```

Description

This function returns the length of the string, which can be zero up to MAXSTRINGLEN.

Name

`function::substr` — Returns a substring.

Synopsis

```
substr:string (str:string, start:long, length:long)
```

Arguments

<i>str</i>	The string to take a substring from
<i>start</i>	Starting position. 0 = start of the string.
<i>length</i>	Length of string to return.

General Syntax

```
substr:string (str:string, start:long, stop:long)
```

Description

Returns the substring of the up to the given length starting at the given start position and ending at given stop position.

Name

`function::stringat` — Returns the char at a given position in the string.

Synopsis

```
stringat:long(str:string, pos:long)
```

Arguments

str The string to fetch the character from.

pos The position to get the character from. 0 = start of the string.

General Syntax

```
stringat:long(str:string, pos:long)
```

Description

This function returns the character at a given position in the string or zero if the string doesn't have as many characters.

Name

function::isinstr — Returns whether a string is a substring of another string.

Synopsis

```
isinstr:long(s1:string, s2:string)
```

Arguments

s1 String to search in.

s2 Substring to find.

General syntax

```
isinstr:long(s1:string, s2:string)
```

Description

This function returns 1 if string *s1* contains *s2*, otherwise zero.

Name

function::text_str — Escape any non-printable chars in a string.

Synopsis

```
text_str:string(input:string)
```

Arguments

input The string to escape.

General Syntax

```
text_str:string (input:string)
```

Description

This function accepts a string argument, and any ASCII characters that are not printable are replaced by the corresponding escape sequence in the returned string.

Name

function::text_strn — Escape any non-printable chars in a string.

Synopsis

```
text_strn:string(input:string, len:long, quoted:long)
```

Arguments

<i>input</i>	The string to escape.
<i>len</i>	Maximum length of string to return. 0 means MAXSTRINGLEN.
<i>quoted</i>	Put double quotes around the string. If input string is truncated it will have “...” after the second quote.

General Syntax

```
text_strn:string(input:string, len:long, quoted:long)
```

Description

This function accepts a string of designated length, and any ASCII characters that are not printable are replaced by the corresponding escape sequence in the returned string.

Name

function::tokenize — Return the next non-empty token in a string.

Synopsis

```
tokenize:string(input:string, delim:string)
```

Arguments

input String to tokenize. If NULL, returns the next non-empty token in the string passed in the previous call to `tokenize`.

delim Token delimiter. Set of characters that delimit the tokens.

General Syntax

```
tokenize:string(input:string, delim:string)
```

Description

This function returns the next non-empty token in the given input string, where the tokens are delimited by characters in the `delim` string. If the input string is non-NULL, it returns the first token. If the input string is NULL, it returns the next token in the string passed in the previous call to `tokenize`. If no delimiter is found, the entire remaining input string is returned. It returns NULL when no more tokens are available.

Name

function::str_replace — str_replace Replaces all instances of a substring with another.

Synopsis

```
str_replace:string(prnt_str:string, srch_str:string, rplc_str:string)
```

Arguments

<i>prnt_str</i>	The string to search and replace in.
<i>srch_str</i>	The substring which is used to search in prnt_str string.
<i>rplc_str</i>	The substring which is used to replace srch_str.

General Syntax

```
str_replace:string(prnt_str:string, srch_str:string, rplc_str:string)
```

Description

This function returns the given string with substrings replaced.

Name

function::strtol — strtol - Convert a string to a long.

Synopsis

```
strtol:long (str:string, base:long)
```

Arguments

str String to convert.

base The base to use

General Syntax

```
strtol:long (str:string, base:long)
```

Description

This function converts the string representation of a number to an integer. The base parameter indicates the number base to assume for the string (eg. 16 for hex, 8 for octal, 2 for binary).

Name

function::isdigit — Checks for a digit.

Synopsis

```
isdigit:long(str:string)
```

Arguments

str String to check.

General Syntax

```
isdigit:long(str:string)
```

Description

Checks for a digit (0 through 9) as the first character of a string. Returns non-zero if true, and a zero if false.

Chapter 19. Utility functions for using ansi control chars in logs

Utility functions for logging using ansi control characters. This lets you manipulate the cursor position and character color output and attributes of log messages.

Name

function::ansi_clear_screen — Move cursor to top left and clear screen.

Synopsis

```
ansi_clear_screen()
```

Arguments

None

General Syntax

```
ansi_clear_screen
```

Description

Sends ansi code for moving cursor to top left and then the ansi code for clearing the screen from the cursor position to the end.

Name

function::ansi_set_color — Set the ansi Select Graphic Rendition mode.

Synopsis

```
ansi_set_color (fg:long)
```

Arguments

fg Foreground color to set.

General Syntax

```
ansi_set_color(fh:long)
```

Description

Sends ansi code for Select Graphic Rendition mode for the given foreground color. Black (30), Blue (34), Green (32), Cyan (36), Red (31), Purple (35), Brown (33), Light Gray (37).

Name

function::ansi_set_color2 — Set the ansi Select Graphic Rendition mode.

Synopsis

```
ansi_set_color2 (fg:long, bg:long)
```

Arguments

fg Foreground color to set.

bg Background color to set.

General Syntax

```
ansi_set_color2(fg:long, bg:long)
```

Description

Sends ansi code for Select Graphic Rendition mode for the given foreground color, Black (30), Blue (34), Green (32), Cyan (36), Red (31), Purple (35), Brown (33), Light Gray (37) and the given background color, Black (40), Red (41), Green (42), Yellow (43), Blue (44), Magenta (45), Cyan (46), White (47).

Name

function::ansi_set_color3 — Set the ansi Select Graphic Rendition mode.

Synopsis

```
ansi_set_color3 (fg:long, bg:long, attr:long)
```

Arguments

fg Foreground color to set.

bg Background color to set.

attr Color attribute to set.

General Syntax

```
ansi_set_color3(fg:long, bg:long, attr:long)
```

Description

Sends ansi code for Select Graphic Rendition mode for the given foreground color, Black (30), Blue (34), Green (32), Cyan (36), Red (31), Purple (35), Brown (33), Light Gray (37), the given background color, Black (40), Red (41), Green (42), Yellow (43), Blue (44), Magenta (45), Cyan (46), White (47) and the color attribute All attributes off (0), Intensity Bold (1), Underline Single (4), Blink Slow (5), Blink Rapid (6), Image Negative (7).

Name

function::ansi_reset_color — Resets Select Graphic Rendition mode.

Synopsis

```
ansi_reset_color()
```

Arguments

None

General Syntax

```
ansi_reset_color
```

Description

Sends ansi code to reset foreground, background and color attribute to default values.

Name

function::ansi_new_line — Move cursor to new line.

Synopsis

```
ansi_new_line()
```

Arguments

None

General Syntax

```
ansi_new_line
```

Description

Sends ansi code new line.

Name

function::ansi_cursor_move — Move cursor to new coordinates.

Synopsis

```
ansi_cursor_move(x:long, y:long)
```

Arguments

x Row to move the cursor to.

y Column to move the cursor to.

General Syntax

```
ansi_curos_move(x:long, y:long)
```

Description

Sends ansi code for positioning the cursor at row *x* and column *y*. Coordinates start at one, (1,1) is the top-left corner.

Name

function::ansi_cursor_hide — Hides the cursor.

Synopsis

```
ansi_cursor_hide()
```

Arguments

None

General Syntax

```
ansi_cusor_hide
```

Description

Sends ansi code for hiding the cursor.

Name

function::ansi_cursor_save — Saves the cursor position.

Synopsis

```
ansi_cursor_save()
```

Arguments

None

General Syntax

```
ansi_cursor_save
```

Description

Sends ansi code for saving the current cursor position.

Name

function::ansi_cursor_restore — Restores a previously saved cursor position.

Synopsis

```
ansi_cursor_restore()
```

Arguments

None

General Syntax

```
ansi_cursor_restore
```

Description

Sends ansi code for restoring the current cursor position previously saved with `ansi_cursor_save`.

Name

function::ansi_cursor_show — Shows the cursor.

Synopsis

```
ansi_cursor_show()
```

Arguments

None

General Syntax

```
ansi_cursor_show
```

Description

Sends ansi code for showing the cursor.