

SystemTap Tapset Reference Manual

SystemTap

SystemTap Tapset Reference Manual

by SystemTap

Copyright © 2008-2010 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::probefunc	34
function::probemod	35
function::modname	36
function::symname	37
function::symdata	38
function::umodname	39
function::usymname	40
function::usymdata	41
function::print_ustack	42
function::sprint_ustack	43
function::print_backtrace	44
function::sprint_backtrace	45
function::backtrace	46
function::task_backtrace	47
function::caller	48
function::caller_addr	49
function::print_ubacktrace	50
function::sprint_ubacktrace	51
function::print_ubacktrace_brief	52
function::ubacktrace	53
function::task_current	54
function::task_parent	55

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

function::proc_mem_string_pid	114
6. Task Time Tapset	115
function::task_utime	116
function::task_utime_tid	117
function::task_stime	118
function::task_stime_tid	119
function::cputime_to_msecs	120
function::msecs_to_string	121
function::cputime_to_string	122
function::task_time_string	123
function::task_time_string_tid	124
7. Scheduler Tapset	125
probe::scheduler.ctxswitch	126
probe::scheduler.kthread_stop	127
probe::scheduler.kthread_stop.return	128
probe::scheduler.wait_task	129
probe::scheduler.wakeup	130
probe::scheduler.wakeup_new	131
probe::scheduler.migrate	132
probe::scheduler.process_free	133
probe::scheduler.process_exit	134
probe::scheduler.process_wait	135
probe::scheduler.process_fork	136
probe::scheduler.signal_send	137
8. IO Scheduler and block IO Tapset	138
probe::ioscheduler.elv_next_request	139
probe::ioscheduler.elv_next_request.return	140
probe::ioscheduler.elv_completed_request	141
probe::ioscheduler.elv_add_request.kp	142
probe::ioscheduler.elv_add_request.tp	143
probe::ioscheduler.elv_add_request	144
probe::ioscheduler_trace.elv_completed_request	145
probe::ioscheduler_trace.elv_issue_request	146
probe::ioscheduler_trace.elv_requeue_request	147
probe::ioscheduler_trace.elv_abort_request	148
probe::ioscheduler_trace.plugin	149
probe::ioscheduler_trace.unplug_io	150
probe::ioscheduler_trace.unplug_timer	151
probe::ioblock.request	152
probe::ioblock.end	153
probe::ioblock_trace.bounce	154
probe::ioblock_trace.request	155
probe::ioblock_trace.end	156
9. SCSI Tapset	157
probe::scsi.ioentry	158
probe::scsi.iodispatching	159
probe::scsi.iodone	160
probe::scsi.iocompleted	161
probe::scsi.ioexecute	162
probe::scsi.set_state	163
10. TTY Tapset	164
probe::tty.open	165
probe::tty.release	166
probe::tty.resize	167
probe::tty.ioctl	168
probe::tty.init	169
probe::tty.register	170
probe::tty.unregister	171

probe::tty.poll	172
probe::tty.receive	173
probe::tty.write	174
probe::tty.read	175
11. Interrupt Request (IRQ) Tapset	176
probe::workqueue.create	177
probe::workqueue.insert	178
probe::workqueue.execute	179
probe::workqueue.destroy	180
probe::irq_handler.entry	181
probe::irq_handler.exit	182
probe::softirq.entry	183
probe::softirq.exit	184
12. Networking Tapset	185
probe::netdev.receive	186
probe::netdev.transmit	187
probe::netdev.change_mtu	188
probe::netdev.open	189
probe::netdev.close	190
probe::netdev.hard_transmit	191
probe::netdev.rx	192
probe::netdev.change_rx_flag	193
probe::netdev.set_promiscuity	194
probe::netdev.ioctl	195
probe::netdev.register	196
probe::netdev.unregister	197
probe::netdev.get_stats	198
probe::netdev.change_mac	199
probe::tcp.sendmsg	200
probe::tcp.sendmsg.return	201
probe::tcp.recvmsg	202
probe::tcp.recvmsg.return	203
probe::tcp.disconnect	204
probe::tcp.disconnect.return	205
probe::tcp.setsockopt	206
probe::tcp.setsockopt.return	207
probe::tcp.receive	208
probe::udp.sendmsg	209
probe::udp.sendmsg.return	210
probe::udp.recvmsg	211
probe::udp.recvmsg.return	212
probe::udp.disconnect	213
probe::udp.disconnect.return	214
function::ip_ntop	215
13. Socket Tapset	216
probe::socket.send	217
probe::socket.receive	218
probe::socket.sendmsg	219
probe::socket.sendmsg.return	220
probe::socket.recvmsg	221
probe::socket.recvmsg.return	222
probe::socket.aio_write	223
probe::socket.aio_write.return	224
probe::socket.aio_read	225
probe::socket.aio_read.return	226
probe::socket.writev	227
probe::socket.writev.return	228
probe::socket.readv	229

probe::socket.readv.return	230
probe::socket.create	231
probe::socket.create.return	232
probe::socket.close	233
probe::socket.close.return	234
function::sock_prot_num2str	235
function::sock_prot_str2num	236
function::sock_fam_num2str	237
function::sock_fam_str2num	238
function::sock_state_num2str	239
function::sock_state_str2num	240
14. SNMP Information Tapset	241
function::ipmib_remote_addr	242
function::ipmib_local_addr	243
function::ipmib_tcp_remote_port	244
function::ipmib_tcp_local_port	245
function::ipmib_get_proto	246
probe::ipmib.InReceives	247
probe::ipmib.InNoRoutes	248
probe::ipmib.InAddrErrors	249
probe::ipmib.InUnknownProtos	250
probe::ipmib.InDiscards	251
probe::ipmib.ForwDatagrams	252
probe::ipmib.OutRequests	253
probe::ipmib.ReasmTimeout	254
probe::ipmib.ReasmReqds	255
probe::ipmib.FragOKs	256
probe::ipmib.FragFails	257
function::tcpmib_get_state	258
function::tcpmib_local_addr	259
function::tcpmib_remote_addr	260
function::tcpmib_local_port	261
function::tcpmib_remote_port	262
probe::tcpmib.ActiveOpens	263
probe::tcpmib.AttemptFails	264
probe::tcpmib.CurrEstab	265
probe::tcpmib.EstabResets	266
probe::tcpmib.InSegs	267
probe::tcpmib.OutRsts	268
probe::tcpmib.OutSegs	269
probe::tcpmib.PassiveOpens	270
probe::tcpmib.RetransSegs	271
probe::linuxmib.DelayedACKs	272
probe::linuxmib.ListenOverflows	273
probe::linuxmib.ListenDrops	274
probe::linuxmib.TCPMemoryPressures	275
15. Kernel Process Tapset	276
probe::kprocess.create	277
probe::kprocess.start	278
probe::kprocess.exec	279
probe::kprocess.exec_complete	280
probe::kprocess.exit	281
probe::kprocess.release	282
16. Signal Tapset	283
probe::signal.send	284
probe::signal.send.return	285
probe::signal.checkperm	286
probe::signal.checkperm.return	287

probe::signal.wakeup	288
probe::signal.check_ignored	289
probe::signal.check_ignored.return	290
probe::signal.force_segv	291
probe::signal.force_segv.return	292
probe::signal.syskill	293
probe::signal.syskill.return	294
probe::signal.sys_tkill	295
probe::signal.systkill.return	296
probe::signal.sys_tgkill	297
probe::signal.sys_tgkill.return	298
probe::signal.send_sig_queue	299
probe::signal.send_sig_queue.return	300
probe::signal.pending	301
probe::signal.pending.return	302
probe::signal.handle	303
probe::signal.handle.return	304
probe::signal.do_action	305
probe::signal.do_action.return	306
probe::signal.procmask	307
probe::signal.procmask.return	308
probe::signal.flush	309
17. Errno Tapset	310
function::errno_str	311
function::returnstr	312
function::return_str	313
18. Directory-entry (dentry) Tapset	314
function::d_name	315
function::inode_name	316
function::reverse_path_walk	317
function::task_dentry_path	318
function::d_path	319
19. Logging Tapset	320
function::log	321
function::warn	322
function::exit	323
function::error	324
function::ftrace	325
20. Queue Statistics Tapset	326
function::qs_wait	327
function::qs_run	328
function::qs_done	329
function::qsq_start	330
function::qsq_utilization	331
function::qsq_blocked	332
function::qsq_wait_queue_length	333
function::qsq_service_time	334
function::qsq_wait_time	335
function::qsq_throughput	336
function::qsq_print	337
21. Random functions Tapset	338
function::randint	339
22. String and data retrieving functions Tapset	340
function::kernel_string	341
function::kernel_string2	342
function::kernel_string_n	343
function::kernel_long	344
function::kernel_int	345

function::kernel_short	346
function::kernel_char	347
function::kernel_pointer	348
function::user_string	349
function::user_string2	350
function::user_string_warn	351
function::user_string_quoted	352
function::user_string_n	353
function::user_string_n2	354
function::user_string_n_warn	355
function::user_string_n_quoted	356
function::user_char	357
function::user_char_warn	358
function::user_short	359
function::user_short_warn	360
function::user_ushort	361
function::user_ushort_warn	362
function::user_int	363
function::user_int_warn	364
function::user_long	365
function::user_long_warn	366
function::user_int8	367
function::user_uint8	368
function::user_int16	369
function::user_uint16	370
function::user_int32	371
function::user_uint32	372
function::user_int64	373
function::user_uint64	374
23. String and data writing functions Tapset	375
function::set_kernel_string	376
function::set_kernel_string_n	377
function::set_kernel_long	378
function::set_kernel_int	379
function::set_kernel_short	380
function::set_kernel_char	381
function::set_kernel_pointer	382
24. A collection of standard string functions	383
function::strlen	384
function::substr	385
function::stringat	386
function::isinstr	387
function::text_str	388
function::text_strn	389
function::str_replace	390
function::strtol	391
function::isdigit	392
function::tokenize	393
25. Utility functions for using ansi control chars in logs	394
function::ansi_clear_screen	395
function::ansi_set_color	396
function::ansi_set_color2	397
function::ansi_set_color3	398
function::ansi_reset_color	399
function::ansi_new_line	400
function::ansi_cursor_move	401
function::ansi_cursor_hide	402
function::ansi_cursor_save	403

function::ansi_cursor_restore	404
function::ansi_cursor_show	405
function::thread_indent	406
function::indent	407
26. SystemTap Translator Tapset	408
probe::stap.pass0	409
probe::stap.pass0.end	410
probe::stap.pass1a	411
probe::stap.pass1b	412
probe::stap.pass1.end	413
probe::stap.pass2	414
probe::stap.pass2.end	415
probe::stap.pass3	416
probe::stap.pass3.end	417
probe::stap.pass4	418
probe::stap.pass4.end	419
probe::stap.pass5	420
probe::stap.pass5.end	421
probe::stap.pass6	422
probe::stap.pass6.end	423
probe::stap.cache_clean	424
probe::stap.cache_add_mod	425
probe::stap.cache_add_src	426
probe::stap.cache_add_nss	427
probe::stap.cache_get	428
probe::stap.system	429
probe::stap.system.spawn	430
probe::stap.system.return	431
probe::staprun.insert_module	432
probe::staprun.remove_module	433
probe::staprun.send_control_message	434
probe::stapio.receive_control_message	435

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Description

This function returns the current cpu number.

Name

function::pp — Returns the active probe point

Synopsis

```
pp:string()
```

Arguments

None

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

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

Description

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

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

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

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

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

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

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

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

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 `symdata`. 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.

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)

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

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

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

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

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

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 to map to a kernel module name

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

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

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::umodname` — Returns the (short) name of the user module. EXPERIMENTAL!

Synopsis

```
umodname:string(addr:long)
```

Arguments

addr User-space address

Description

Returns the short name of the user space module for the current task that that the given address is part of. Returns “<unknown>” when the address isn’t in a (mapped in) module, or the module cannot be found for some reason.

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

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:string()
```

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

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`

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

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

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. Returns nothing.

Note

To get (full) backtraces for user space applications and shared libraries not mentioned in the current script run `stap` with `-d /path/to/exe-or-so` and/or add `--ldd` to load all needed unwind data.

Name

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

Synopsis

```
sprint_ubacktrace:string()
```

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).

Note

To get (full) backtraces for user space applications and shared shared libraries not mentioned in the current script run stap with `-d /path/to/exe-or-so` and/or add `--ldd` to load all needed unwind data.

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).

Note

To get (full) backtraces for user space applications and shared libraries not mentioned in the current script run `stap` with `-d /path/to/exe-or-so` and/or add `--ldd` to load all needed unwind data.

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.

Note

To get (full) backtraces for user space applications and shared libraries not mentioned in the current script run stap with `-d /path/to/exe-or-so` and/or add `--ldd` to load all needed unwind data.

Name

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

Synopsis

```
task_current:long()
```

Arguments

None

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

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

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), or `EXIT_DEAD` (32).

Name

function::task_execname — The name of the task

Synopsis

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

Arguments

task task_struct pointer

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Description

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

Chapter 4. Time utility functions

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

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`)

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.

Name

function::tz_gmtoff — Return local time zone offset

Synopsis

```
tz_gmtoff()
```

Arguments

None

Description

Returns the local time zone offset (seconds west of UTC), as passed by staprun at script startup only.

Name

function::tz_name — Return local time zone name

Synopsis

```
tz_name ()
```

Arguments

None

Description

Returns the local time zone name, as passed by staprun at script startup only.

Name

function::tz_ctime — Convert seconds since epoch into human readable date/time string, with local time zone

Synopsis

```
tz_ctime(epochsecs:)
```

Arguments

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

Description

Takes an argument of seconds since the epoch as returned by `gettimeofday_s`. Returns a string of the same form as `ctime`, but offsets the epoch time for the local time zone, and appends the name of the local time zone. The string length may vary. The time zone information is passed by `staprun` at script startup only.

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

name name of the probe point

fault_type 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

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 kmem_cache_alloc is requested

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

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 kmem_cache_alloc_node is requested

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

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 kmem_cache_free is requested

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

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. Scheduler Tapset

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

Name

probe::scheduler.ctxswitch — A context switch is occurring.

Synopsis

```
scheduler.ctxswitch
```

Values

<i>prev_pid</i>	The PID of the process to be switched out
<i>name</i>	name of the probe point
<i>next_task_name</i>	The name of the process to be switched in
<i>nexttsk_state</i>	the state of the process to be switched in
<i>prev_priority</i>	The priority of the process to be switched out
<i>next_pid</i>	The PID of the process to be switched in
<i>next_priority</i>	The priority of the process to be switched in
<i>prevtsk_state</i>	the state of the process to be switched out
<i>next_tid</i>	The TID of the process to be switched in
<i>prev_task_name</i>	The name of the process to be switched out
<i>prev_tid</i>	The TID of the process to be switched out

Description

Currently, SystemTap can't access arguments of inline functions. So we choose to probe `__switch_to` instead of `context_switch`

Name

probe::scheduler.kthread_stop — A thread created by kthread_create is being stopped

Synopsis

```
scheduler.kthread_stop
```

Values

<i>thread_priority</i>	priority of the thread
<i>thread_pid</i>	PID of the thread being stopped

Name

probe::scheduler.kthread_stop.return — A kthread is stopped and gets the return value

Synopsis

```
scheduler.kthread_stop.return
```

Values

<i>return_value</i>	return value after stopping the thread
<i>name</i>	name of the probe point

Name

probe::scheduler.wait_task — Waiting on a task to unschedule (become inactive)

Synopsis

```
scheduler.wait_task
```

Values

<i>name</i>	name of the probe point
<i>task_pid</i>	PID of the task the scheduler is waiting on
<i>task_priority</i>	priority of the task

Name

probe::scheduler.wakeup — Task is woken up

Synopsis

`scheduler.wakeup`

Values

<i>task_cpu</i>	cpu of the task being woken up
<i>name</i>	name of the probe point
<i>task_pid</i>	PID of the task being woken up
<i>task_priority</i>	priority of the task being woken up
<i>task_state</i>	state of the task being woken up
<i>task_tid</i>	tid of the task being woken up

Name

probe::scheduler.wakeup_new — Newly created task is woken up for the first time

Synopsis

```
scheduler.wakeup_new
```

Values

<i>task_cpu</i>	cpu of the task woken up
<i>name</i>	name of the probe point
<i>task_pid</i>	PID of the new task woken up
<i>task_priority</i>	priority of the new task
<i>task_state</i>	state of the task woken up
<i>task_tid</i>	TID of the new task woken up

Name

probe::scheduler.migrate — Task migrating across cpus

Synopsis

```
scheduler.migrate
```

Values

<i>priority</i>	priority of the task being migrated
<i>cpu_from</i>	the original cpu
<i>name</i>	name of the probe point
<i>task</i>	the process that is being migrated
<i>cpu_to</i>	the destination cpu
<i>pid</i>	PID of the task being migrated

Name

probe::scheduler.process_free — Scheduler freeing a data structure for a process

Synopsis

```
scheduler.process_free
```

Values

<i>priority</i>	priority of the process getting freed
<i>name</i>	name of the probe point
<i>pid</i>	PID of the process getting freed

Name

probe::scheduler.process_exit — Process exiting

Synopsis

```
scheduler.process_exit
```

Values

<i>priority</i>	priority of the process exiting
<i>name</i>	name of the probe point
<i>pid</i>	PID of the process exiting

Name

probe::scheduler.process_wait — Scheduler starting to wait on a process

Synopsis

```
scheduler.process_wait
```

Values

<i>name</i>	name of the probe point
<i>pid</i>	PID of the process scheduler is waiting on

Name

probe::scheduler.process_fork — Process forked

Synopsis

```
scheduler.process_fork
```

Values

<i>name</i>	name of the probe point
<i>parent_pid</i>	PID of the parent process
<i>child_pid</i>	PID of the child process

Name

probe::scheduler.signal_send — Sending a signal

Synopsis

```
scheduler.signal_send
```

Values

<i>signal_number</i>	signal number
<i>name</i>	name of the probe point
<i>pid</i>	pid of the process sending signal

Chapter 8. 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 9. 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 10. 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 11. Interrupt Request (IRQ) Tapset

This family of probe points is used to probe interrupt request (IRQ) activities. It contains the following probe points:

Name

probe::workqueue.create — Creating a new workqueue

Synopsis

```
workqueue.create
```

Values

<i>wq_thread</i>	task_struct of the workqueue thread
<i>cpu</i>	cpu for which the worker thread is created

Name

probe::workqueue.insert — Queuing work on a workqueue

Synopsis

```
workqueue.insert
```

Values

<i>wq_thread</i>	task_struct of the workqueue thread
<i>work_func</i>	pointer to handler function
<i>work</i>	work_struct* being queued

Name

probe::workqueue.execute — Executing deferred work

Synopsis

```
workqueue.execute
```

Values

<i>wq_thread</i>	task_struct of the workqueue thread
<i>work_func</i>	pointer to handler function
<i>work</i>	work_struct* being executed

Name

probe::workqueue.destroy — Destroying workqueue

Synopsis

```
workqueue.destroy
```

Values

<i>wq_thread</i>	task_struct of the workqueue thread
------------------	-------------------------------------

Name

probe::irq_handler.entry — Execution of interrupt handler starting

Synopsis

```
irq_handler.entry
```

Values

<i>dev_name</i>	name of device
<i>flags</i>	Flags for IRQ handler
<i>dev_id</i>	Cookie to identify device
<i>dir</i>	pointer to the proc/irq/NN/name entry
<i>irq</i>	irq number
<i>next_irqaction</i>	pointer to next irqaction for shared interrupts
<i>thread_flags</i>	Flags related to thread
<i>thread</i>	thread pointer for threaded interrupts
<i>thread_fn</i>	interrupt handler function for threaded interrupts
<i>handler</i>	interrupt handler function
<i>flags_str</i>	symbolic string representation of IRQ flags
<i>action</i>	struct irqaction* for this interrupt num

Name

probe::irq_handler.exit — Execution of interrupt handler completed

Synopsis

```
irq_handler.exit
```

Values

<i>dev_name</i>	name of device
<i>ret</i>	return value of the handler
<i>flags</i>	flags for IRQ handler
<i>dev_id</i>	Cookie to identify device
<i>dir</i>	pointer to the proc/irq/NN/name entry
<i>next_irqaction</i>	pointer to next irqaction for shared interrupts
<i>irq</i>	interrupt number
<i>thread_flags</i>	Flags related to thread
<i>thread</i>	thread pointer for threaded interrupts
<i>thread_fn</i>	interrupt handler function for threaded interrupts
<i>flags_str</i>	symbolic string representation of IRQ flags
<i>handler</i>	interrupt handler function that was executed
<i>action</i>	struct irqaction*

Name

probe::softirq.entry — Execution of handler for a pending softirq starting

Synopsis

```
softirq.entry
```

Values

<i>vec</i>	softirq_action vector
<i>h</i>	struct softirq_action* for current pending softirq
<i>vec_nr</i>	softirq vector number
<i>action</i>	pointer to softirq handler just about to execute

Name

probe::softirq.exit — Execution of handler for a pending softirq completed

Synopsis

```
softirq.exit
```

Values

<i>vec</i>	softirq_action vector
<i>h</i>	struct softirq_action* for just executed softirq
<i>vec_nr</i>	softirq vector number
<i>action</i>	pointer to softirq handler that just finished execution

Chapter 12. 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 13. 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 14. SNMP Information Tapset

This family of probe points is used to probe socket activities to provide SNMP type information. It contains the following functions and probe points:

Name

function::ipmib_remote_addr — Get the remote ip address

Synopsis

```
ipmib_remote_addr:long(skb:long,SourceIsLocal:long)
```

Arguments

<i>skb</i>	pointer to a struct sk_buff
<i>SourceIsLocal</i>	flag to indicate whether local operation

Description

Returns the remote ip address from *skb*.

Name

function::ipmib_local_addr — Get the local ip address

Synopsis

```
ipmib_local_addr:long(skb:long, SourceIsLocal:long)
```

Arguments

<i>skb</i>	pointer to a struct <code>sk_buff</code>
<i>SourceIsLocal</i>	flag to indicate whether local operation

Description

Returns the local ip address *skb*.

Name

function::ipmib_tcp_remote_port — Get the remote tcp port

Synopsis

```
ipmib_tcp_remote_port:long(skb:long,SourceIsLocal:long)
```

Arguments

<i>skb</i>	pointer to a struct sk_buff
<i>SourceIsLocal</i>	flag to indicate whether local operation

Description

Returns the remote tcp port from *skb*.

Name

function::ipmib_tcp_local_port — Get the local tcp port

Synopsis

```
ipmib_tcp_local_port:long(skb:long,SourceIsLocal:long)
```

Arguments

<i>skb</i>	pointer to a struct <code>sk_buff</code>
<i>SourceIsLocal</i>	flag to indicate whether local operation

Description

Returns the local tcp port from *skb*.

Name

function::ipmib_get_proto — Get the protocol value

Synopsis

```
ipmib_get_proto:long(skb:long)
```

Arguments

skb pointer to a struct sk_buff

Description

Returns the protocol value from *skb*.

Name

probe::ipmib.InReceives — Count an arriving packet

Synopsis

```
ipmib.InReceives
```

Values

skb pointer to the struct `sk_buff` being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `ipmib_filter_key`. If the packet passes the filter is counted in the global *InReceives* (equivalent to SNMP's MIB IP-STATS_MIB_INRECEIVES)

Name

probe::ipmib.InNoRoutes — Count an arriving packet with no matching socket

Synopsis

```
ipmib.InNoRoutes
```

Values

skb pointer to the struct `sk_buff` being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `ipmib_filter_key`. If the packet passes the filter is counted in the global *InNoRoutes* (equivalent to SNMP's MIB IP-STATS-MIB-INNOROUTES)

Name

probe::ipmib.InAddrErrors — Count arriving packets with an incorrect address

Synopsis

```
ipmib.InAddrErrors
```

Values

skb pointer to the struct `sk_buff` being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `ipmib_filter_key`. If the packet passes the filter is counted in the global *InAddrErrors* (equivalent to SNMP's MIB IP-STATS_MIB_INADDRERRORS)

Name

probe::ipmib.InUnknownProtos — Count arriving packets with an unbound proto

Synopsis

```
ipmib.InUnknownProtos
```

Values

skb pointer to the struct *sk_buff* being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `ipmib_filter_key`. If the packet passes the filter is counted in the global *InUnknownProtos* (equivalent to SNMP's MIB IPSTATS_MIB_INUNKNOWNPROTOS)

Name

probe::ipmib.InDiscards — Count discarded inbound packets

Synopsis

```
ipmib.InDiscards
```

Values

skb pointer to the struct `sk_buff` being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `ipmib_filter_key`. If the packet passes the filter is counted in the global *InDiscards* (equivalent to SNMP's MIB `STATS_MIB_INDISCARDS`)

Name

probe::ipmib.ForwDatagrams — Count forwarded packet

Synopsis

```
ipmib.ForwDatagrams
```

Values

skb pointer to the struct `sk_buff` being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `ipmib_filter_key`. If the packet passes the filter is counted in the global *ForwDatagrams* (equivalent to SNMP's MIB IP-STATS_MIB_OUTFORWDATAGRAMS)

Name

probe::ipmib.OutRequests — Count a request to send a packet

Synopsis

```
ipmib.OutRequests
```

Values

skb pointer to the struct `sk_buff` being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `ipmib_filter_key`. If the packet passes the filter is counted in the global *OutRequests* (equivalent to SNMP's MIB IP-STATS_MIB_OUTREQUESTS)

Name

probe::ipmib.ReasmTimeout — Count Reassembly Timeouts

Synopsis

```
ipmib.ReasmTimeout
```

Values

skb pointer to the struct *sk_buff* being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `ipmib_filter_key`. If the packet passes the filter is is counted in the global *ReasmTimeout* (equivalent to SNMP's MIB IP-STATS_MIB_REASMTIMEOUT)

Name

probe::ipmib.ReasmReqds — Count number of packet fragments reassembly requests

Synopsis

```
ipmib.ReasmReqds
```

Values

skb pointer to the struct `sk_buff` being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `ipmib_filter_key`. If the packet passes the filter is counted in the global *ReasmReqds* (equivalent to SNMP's MIB IP-STATS_MIB_REASMREQDS)

Name

probe::ipmib.FragOKs — Count datagram fragmented successfully

Synopsis

```
ipmib.FragOKs
```

Values

skb pointer to the struct `sk_buff` being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `ipmib_filter_key`. If the packet passes the filter is counted in the global *FragOKs* (equivalent to SNMP's MIB IP-STATS_MIB_FRAGOKS)

Name

probe::ipmib.FragFails — Count datagram fragmented unsuccessfully

Synopsis

```
ipmib.FragFails
```

Values

skb pointer to the struct `sk_buff` being acted on

op Value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `ipmib_filter_key`. If the packet passes the filter is counted in the global *FragFails* (equivalent to SNMP's MIB `IPSTATS_MIB_FRAGFAILS`)

Name

function::tcpmib_get_state — Get a socket's state

Synopsis

```
tcpmib_get_state:long(sk:long)
```

Arguments

sk pointer to a struct sock

Description

Returns the sk_state from a struct sock.

Name

function::tcpmib_local_addr — Get the source address

Synopsis

```
tcpmib_local_addr:long(sk:long)
```

Arguments

sk pointer to a struct inet_sock

Description

Returns the saddr from a struct inet_sock in host order.

Name

function::tcpmib_remote_addr — Get the remote address

Synopsis

```
tcpmib_remote_addr:long(sk:long)
```

Arguments

sk pointer to a struct inet_sock

Description

Returns the daddr from a struct inet_sock in host order.

Name

function::tcpmib_local_port — Get the local port

Synopsis

```
tcpmib_local_port:long(sk:long)
```

Arguments

sk pointer to a struct inet_sock

Description

Returns the sport from a struct inet_sock in host order.

Name

function::tcpmib_remote_port — Get the remote port

Synopsis

```
tcpmib_remote_port:long(sk:long)
```

Arguments

sk pointer to a struct inet_sock

Description

Returns the dport from a struct inet_sock in host order.

Name

probe::tcpmib.ActiveOpens — Count an active opening of a socket

Synopsis

```
tcpmib.ActiveOpens
```

Values

sk pointer to the struct sock being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `tcpmib_filter_key`. If the packet passes the filter is counted in the global *ActiveOpens* (equivalent to SNMP's MIB TCP_MIB_ACTIVEOPENS)

Name

probe::tcpmib.AttemptFails — Count a failed attempt to open a socket

Synopsis

```
tcpmib.AttemptFails
```

Values

skb pointer to the struct sock being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `tcpmib_filter_key`. If the packet passes the filter is counted in the global *AttemptFails* (equivalent to SNMP's MIB TCP_MIB_ATTEMPTFAILS)

Name

probe::tcpmib.CurrEstab — Update the count of open sockets

Synopsis

```
tcpmib.CurrEstab
```

Values

skb pointer to the struct sock being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `tcpmib_filter_key`. If the packet passes the filter is counted in the global *CurrEstab* (equivalent to SNMP's MIB TCP_MIB_CURRESTAB)

Name

probe::tcpmib.EstabResets — Count the reset of a socket

Synopsis

```
tcpmib.EstabResets
```

Values

sk pointer to the struct sock being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `tcpmib_filter_key`. If the packet passes the filter is counted in the global *EstabResets* (equivalent to SNMP's MIB TCP_MIB_ESTABRESETS)

Name

probe::tcpmib.InSegs — Count an incoming tcp segment

Synopsis

```
tcpmib.InSegs
```

Values

sk pointer to the struct sock being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `tcpmib_filter_key` (or `ipmib_filter_key` for tcp v4). If the packet passes the filter is counted in the global *InSegs* (equivalent to SNMP's MIB TCP_MIB_INSEGS)

Name

probe::tcpmib.OutRsts — Count the sending of a reset packet

Synopsis

```
tcpmib.OutRsts
```

Values

skb pointer to the struct sock being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `tcpmib_filter_key`. If the packet passes the filter is counted in the global *OutRsts* (equivalent to SNMP's MIB TCP_MIB_OUTRSTS)

Name

probe::tcpmib.OutSegs — Count the sending of a TCP segment

Synopsis

```
tcpmib.OutSegs
```

Values

skb pointer to the struct sock being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `tcpmib_filter_key`. If the packet passes the filter is counted in the global *OutSegs* (equivalent to SNMP's MIB TCP_MIB_OUTSEGS)

Name

probe::tcpmib.PassiveOpens — Count the passive creation of a socket

Synopsis

```
tcpmib.PassiveOpens
```

Values

sk pointer to the struct sock being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `tcpmib_filter_key`. If the packet passes the filter is counted in the global *PassiveOpens* (equivalent to SNMP's MIB TCP_MIB_PASSIVEOPENS)

Name

probe::tcpmib.RetransSegs — Count the retransmission of a TCP segment

Synopsis

```
tcpmib.RetransSegs
```

Values

sk pointer to the struct sock being acted on

op value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `tcpmib_filter_key`. If the packet passes the filter is counted in the global *RetransSegs* (equivalent to SNMP's MIB TCP_MIB_RETRANSSEGS)

Name

probe::linuxmib.DelayedACKs — Count of delayed acks

Synopsis

```
linuxmib.DelayedACKs
```

Values

sk Pointer to the struct sock being acted on

op Value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `linuxmib_filter_key`. If the packet passes the filter is counted in the global *DelayedACKs* (equivalent to SNMP's MIB LINUX_MIB_DELAYEDACKS)

Name

probe::linuxmib.ListenOverflows — Count of times a listen queue overflowed

Synopsis

```
linuxmib.ListenOverflows
```

Values

sk Pointer to the struct sock being acted on

op Value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `linuxmib_filter_key`. If the packet passes the filter is counted in the global *ListenOverflows* (equivalent to SNMP's MIB LINUX_MIB_LISTENOVERFLOWS)

Name

probe::linuxmib.ListenDrops — Count of times conn request that were dropped

Synopsis

```
linuxmib.ListenDrops
```

Values

sk Pointer to the struct sock being acted on

op Value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `linuxmib_filter_key`. If the packet passes the filter is counted in the global *ListenDrops* (equivalent to SNMP's MIB LINUX_MIB_LISTENDROPS)

Name

probe::linuxmib.TCPMemoryPressures — Count of times memory pressure was used

Synopsis

linuxmib.TCPMemoryPressures

Values

skb Pointer to the struct sock being acted on

op Value to be added to the counter (default value of 1)

Description

The packet pointed to by *skb* is filtered by the function `linuxmib_filter_key`. If the packet passes the filter is counted in the global *TCPMemoryPressures* (equivalent to SNMP's MIB LINUX_MIB_TCPMEMORYPRESSURES)

Chapter 15. 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 16. 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 17. Errno Tapset

This set of functions is used to handle errno number values. It contains the following function:

Name

`function::errno_str` — Symbolic string associated with error code

Synopsis

```
errno_str:string(err:long)
```

Arguments

err The error number recieved

Description

This function returns the symbolic string associated with the giver error code, such as ENOENT for the number 2, or E#3333 for an out-of-range value such as 3333.

Name

`function::returnstr` — Formats the return value as a string

Synopsis

```
returnstr:string(format:long)
```

Arguments

format Variable to determine return type base value

Description

This function is used by the `nd_syscall` tapset, and returns a string. Set `format` equal to 1 for a decimal, 2 for hex, 3 for octal.

Note that this function should only be used in dwarfless probes (i.e. `'kprobe.function("foo")'`). Other probes should use `return_str`.

Name

`function::return_str` — Formats the return value as a string

Synopsis

```
return_str:string(format:long,ret:long)
```

Arguments

<i>format</i>	Variable to determine return type base value
<i>ret</i>	Return value (typically <code>\$return</code>)

Description

This function is used by the syscall tapset, and returns a string. Set `format` equal to 1 for a decimal, 2 for hex, 3 for octal.

Note that this function is preferred over `returnstr`.

Chapter 18. 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::inode_name — get the inode name

Synopsis

```
inode_name:string(inode:long)
```

Arguments

inode Pointer to inode.

Description

Returns the first path basename associated with the given inode.

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::task_dentry_path — get the full dentry path

Synopsis

```
task_dentry_path:string(task:long,dentry:long,vfsmnt:long)
```

Arguments

<i>task</i>	task_struct pointer.
<i>dentry</i>	dirent pointer.
<i>vfsmnt</i>	vfsmnt pointer.

Description

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

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 19. 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

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

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

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 20. Queue Statistics Tapset

This family of functions is used to track performance of queuing systems.

Name

function::qs_wait — Function to record enqueue requests

Synopsis

```
qs_wait(qname:string)
```

Arguments

qname the name of the queue requesting enqueue

Description

This function records that a new request was enqueued for the given queue name.

Name

function::qs_run — Function to record being moved from wait queue to being serviced

Synopsis

```
qs_run(qname:string)
```

Arguments

qname the name of the service being moved and started

Description

This function records that the previous enqueued request was removed from the given wait queue and is now being serviced.

Name

function::qs_done — Function to record finishing request

Synopsis

```
qs_done(qname:string)
```

Arguments

qname the name of the service that finished

Description

This function records that a request originally from the given queue has completed being serviced.

Name

function::qsq_start — Function to reset the stats for a queue

Synopsis

```
qsq_start (qname:string)
```

Arguments

qname the name of the service that finished

Description

This function resets the statistics counters for the given queue, and restarts tracking from the moment the function was called. This function is also used to create initialize a queue.

Name

function::qsq_utilization — Fraction of time that any request was being serviced

Synopsis

```
qsq_utilization:long(qname:string, scale:long)
```

Arguments

qname queue name

scale scale variable to take account for interval fraction

Description

This function returns the average time in microseconds that at least one request was being serviced.

Name

function::qsq_blocked — Returns the time request was on the wait queue

Synopsis

```
qsq_blocked:long (qname:string, scale:long)
```

Arguments

qname queue name

scale scale variable to take account for interval fraction

Description

This function returns the fraction of elapsed time during which one or more requests were on the wait queue.

Name

function::qsq_wait_queue_length — length of wait queue

Synopsis

```
qsq_wait_queue_length:long (qname:string, scale:long)
```

Arguments

qname queue name

scale scale variable to take account for interval fraction

Description

This function returns the average length of the wait queue

Name

function::qsq_service_time — Amount of time per request service

Synopsis

```
qsq_service_time:long(qname:string,scale:long)
```

Arguments

qname queue name

scale scale variable to take account for interval fraction

Description

This function returns the average time in microseconds required to service a request once it is removed from the wait queue.

Name

function::qsq_wait_time — Amount of time in queue + service per request

Synopsis

```
qsq_wait_time:long(qname:string, scale:long)
```

Arguments

qname queue name

scale scale variable to take account for interval fraction

Description

This function returns the average time in microseconds that it took for a request to be serviced (qs_wait to qa_done).

Name

function::qsq_throughput — Number of requests served per unit time

Synopsis

```
qsq_throughput:long(qname:string,scale:long)
```

Arguments

qname queue name

scale scale variable to take account for interval fraction

Description

This function returns the average number of requests served per microsecond.

Name

function::qsq_print — Returns a line of statistics for the given queue

Synopsis

```
qsq_print (qname:string)
```

Arguments

qname queue name

Description

This function prints a line containing the following

statistics for the given queue

the queue name, the average rate of requests per second, the average wait queue length, the average time on the wait queue, the average time to service a request, the percentage of time the wait queue was used, and the percentage of time request was being serviced.

Chapter 21. 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 22. 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

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

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

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)

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

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

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

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

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

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

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

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

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)

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

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)

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)

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_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

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

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.

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

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

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_ushort` — Retrieves an unsigned short value stored in user space

Synopsis

```
user_ushort:long(addr:long)
```

Arguments

addr the user space address to retrieve the unsigned short from

Description

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

Name

function::user_ushort_warn — Retrieves an unsigned short value stored in user space

Synopsis

```
user_ushort_warn:long(addr:long)
```

Arguments

addr the user space address to retrieve the unsigned short from

Description

Returns the unsigned 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

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

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

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

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_int8` — Retrieves a 8-bit integer value stored in user space

Synopsis

```
user_int8:long(addr:long)
```

Arguments

addr the user space address to retrieve the 8-bit integer from

Description

Returns the 8-bit integer value from a given user space address. Returns zero when user space data is not accessible.

Name

`function::user_uint8` — Retrieves an unsigned 8-bit integer value stored in user space

Synopsis

```
user_uint8:long(addr:long)
```

Arguments

addr the user space address to retrieve the unsigned 8-bit integer from

Description

Returns the unsigned 8-bit integer value from a given user space address. Returns zero when user space data is not accessible.

Name

`function::user_int16` — Retrieves a 16-bit integer value stored in user space

Synopsis

```
user_int16:long(addr:long)
```

Arguments

addr the user space address to retrieve the 16-bit integer from

Description

Returns the 16-bit integer value from a given user space address. Returns zero when user space data is not accessible.

Name

`function::user_uint16` — Retrieves an unsigned 16-bit integer value stored in user space

Synopsis

```
user_uint16:long(addr:long)
```

Arguments

addr the user space address to retrieve the unsigned 16-bit integer from

Description

Returns the unsigned 16-bit integer value from a given user space address. Returns zero when user space data is not accessible.

Name

`function::user_int32` — Retrieves a 32-bit integer value stored in user space

Synopsis

```
user_int32:long(addr:long)
```

Arguments

addr the user space address to retrieve the 32-bit integer from

Description

Returns the 32-bit integer value from a given user space address. Returns zero when user space data is not accessible.

Name

`function::user_uint32` — Retrieves an unsigned 32-bit integer value stored in user space

Synopsis

```
user_uint32:long(addr:long)
```

Arguments

addr the user space address to retrieve the unsigned 32-bit integer from

Description

Returns the unsigned 32-bit integer value from a given user space address. Returns zero when user space data is not accessible.

Name

`function::user_int64` — Retrieves a 64-bit integer value stored in user space

Synopsis

```
user_int64:long(addr:long)
```

Arguments

addr the user space address to retrieve the 64-bit integer from

Description

Returns the 64-bit integer value from a given user space address. Returns zero when user space data is not accessible.

Name

`function::user_uint64` — Retrieves an unsigned 64-bit integer value stored in user space

Synopsis

```
user_uint64:long(addr:long)
```

Arguments

addr the user space address to retrieve the unsigned 64-bit integer from

Description

Returns the unsigned 64-bit integer value from a given user space address. Returns zero when user space data is not accessible.

Chapter 23. String and data writing functions Tapset

The SystemTap guru mode can be used to test error handling in kernel code by simulating faults. The functions in the this tapset provide standard methods of writing to primitive types in the kernel's memory. All the functions in this tapset require the use of guru mode (**-g**).

Name

`function::set_kernel_string` — Writes a string to kernel memory

Synopsis

```
set_kernel_string(addr:long, val:string)
```

Arguments

addr The kernel address to write the string to

val The string which is to be written

Description

Writes the given string to a given kernel memory address. Reports an error on string copy fault. Requires the use of guru mode (-g).

Name

function::set_kernel_string_n — Writes a string of given length to kernel memory

Synopsis

```
set_kernel_string_n(addr:long,n:long,val:string)
```

Arguments

addr The kernel address to write the string to

n The maximum length of the string

val The string which is to be written

Description

Writes the given string up to a maximum given length to a given kernel memory address. Reports an error on string copy fault. Requires the use of guru mode (-g).

Name

`function::set_kernel_long` — Writes a long value to kernel memory

Synopsis

```
set_kernel_long(addr:long, val:long)
```

Arguments

addr The kernel address to write the long to

val The long which is to be written

Description

Writes the long value to a given kernel memory address. Reports an error when writing to the given address fails. Requires the use of guru mode (-g).

Name

function::set_kernel_int — Writes an int value to kernel memory

Synopsis

```
set_kernel_int (addr:long, val:long)
```

Arguments

addr The kernel address to write the int to

val The int which is to be written

Description

Writes the int value to a given kernel memory address. Reports an error when writing to the given address fails. Requires the use of guru mode (-g).

Name

function::set_kernel_short — Writes a short value to kernel memory

Synopsis

```
set_kernel_short (addr:long, val:long)
```

Arguments

addr The kernel address to write the short to

val The short which is to be written

Description

Writes the short value to a given kernel memory address. Reports an error when writing to the given address fails. Requires the use of guru mode (-g).

Name

function::set_kernel_char — Writes a char value to kernel memory

Synopsis

```
set_kernel_char (addr:long, val:long)
```

Arguments

addr The kernel address to write the char to

val The char which is to be written

Description

Writes the char value to a given kernel memory address. Reports an error when writing to the given address fails. Requires the use of guru mode (-g).

Name

function::set_kernel_pointer — Writes a pointer value to kernel memory.

Synopsis

```
set_kernel_pointer(addr:long, val:long)
```

Arguments

addr The kernel address to write the pointer to

val The pointer which is to be written

Description

Writes the pointer value to a given kernel memory address. Reports an error when writing to the given address fails. Requires the use of guru mode (-g).

Chapter 24. 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

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 of the extracted string (first character is 0)
<i>length</i>	length of string to return

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 (first character is 0)

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

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

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 implies MAXSTRINGLEN)
<i>quoted</i>	put double quotes around the string. If input string is truncated it will have “...” after the second quote

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::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 <i>prnt_str</i> string
<i>rplc_str</i>	the substring which is used to replace <i>srch_str</i>

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

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

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.

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 set of characters that delimit the tokens

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.

Chapter 25. 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

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.

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.

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.

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

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

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.

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

Description

Sends ansi code for hiding the cursor.

Name

function::ansi_cursor_save — Saves the cursor position.

Synopsis

```
ansi_cursor_save()
```

Arguments

None

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

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

Description

Sends ansi code for showing the cursor.

Name

`function::thread_indent` — returns an amount of space with the current task information

Synopsis

```
thread_indent:string(delta:long)
```

Arguments

delta the amount of space added/removed for each call

Description

This function returns a string with appropriate indentation for a thread. Call it with a small positive or matching negative delta. If this is the real outermost, initial level of indentation, then the function resets the relative timestamp base to zero. An example is shown at the end of this file.

Name

`function::indent` — returns an amount of space to indent

Synopsis

```
indent:string(delta:long)
```

Arguments

delta the amount of space added/removed for each call

Description

This function returns a string with appropriate indentation. Call it with a small positive or matching negative delta. Unlike the `thread_indent` function, the `indent` does not track individual indent values on a per thread basis.

Chapter 26. SystemTap Translator Tapset

This family of user-space probe points is used to probe the operation of the SystemTap translator (**stap**) and run command (**staprun**). The tapset includes probes to watch the various phases of SystemTap and SystemTap's management of instrumentation cache. It contains the following probe points:

Name

probe::stap.pass0 — Starting stap pass0 (parsing command line arguments)

Synopsis

```
stap.pass0
```

Values

session the systemtap_session variable s

Description

pass0 fires after command line arguments have been parsed.

Name

probe::stap.pass0.end — Finished stap pass0 (parsing command line arguments)

Synopsis

```
stap.pass0.end
```

Values

session the systemtap_session variable *s*

Description

pass0.end fires just before the `gettimeofday` call for pass1.

Name

probe::stap.pass1a — Starting stap pass1 (parsing user script)

Synopsis

```
stap.pass1a
```

Values

session the systemtap_session variable s

Description

pass1a fires just after the call to `gettimeofday`, before the user script is parsed.

Name

probe::stap.pass1b — Starting stap pass1 (parsing library scripts)

Synopsis

```
stap.pass1b
```

Values

session the systemtap_session variable s

Description

pass1b fires just before the library scripts are parsed.

Name

probe::stap.pass1.end — Finished stap pass1 (parsing scripts)

Synopsis

```
stap.pass1.end
```

Values

session the systemtap_session variable s

Description

pass1.end fires just before the jump to cleanup if s.last_pass = 1.

Name

probe::stap.pass2 — Starting stap pass2 (elaboration)

Synopsis

`stap.pass2`

Values

session the systemtap_session variable *s*

Description

pass2 fires just after the call to `gettimeofday`, just before the call to `semantic_pass`.

Name

probe::stap.pass2.end — Finished stap pass2 (elaboration)

Synopsis

```
stap.pass2.end
```

Values

session the systemtap_session variable *s*

Description

pass2.end fires just before the jump to cleanup if *s*.last_pass = 2

Name

probe::stap.pass3 — Starting stap pass3 (translation to C)

Synopsis

```
stap.pass3
```

Values

session the systemtap_session variable *s*

Description

pass3 fires just after the call to `gettimeofday`, just before the call to `translate_pass`.

Name

probe::stap.pass3.end — Finished stap pass3 (translation to C)

Synopsis

```
stap.pass3.end
```

Values

session the systemtap_session variable s

Description

pass3.end fires just before the jump to cleanup if s.last_pass = 3

Name

probe::stap.pass4 — Starting stap pass4 (compile C code into kernel module)

Synopsis

```
stap.pass4
```

Values

session the systemtap_session variable *s*

Description

pass4 fires just after the call to `gettimeofday`, just before the call to `compile_pass`.

Name

probe::stap.pass4.end — Finished stap pass4 (compile C code into kernel module)

Synopsis

```
stap.pass4.end
```

Values

session the systemtap_session variable s

Description

pass4.end fires just before the jump to cleanup if s.last_pass = 4

Name

probe::stap.pass5 — Starting stap pass5 (running the instrumentation)

Synopsis

```
stap.pass5
```

Values

session the systemtap_session variable *s*

Description

pass5 fires just after the call to `gettimeofday`, just before the call to `run_pass`.

Name

probe::stap.pass5.end — Finished stap pass5 (running the instrumentation)

Synopsis

```
stap.pass5.end
```

Values

session the systemtap_session variable *s*

Description

pass5.end fires just before the cleanup label

Name

probe::stap.pass6 — Starting stap pass6 (cleanup)

Synopsis

`stap.pass6`

Values

session the systemtap_session variable *s*

Description

pass6 fires just after the cleanup label, essentially the same spot as pass5.end

Name

probe::stap.pass6.end — Finished stap pass6 (cleanup)

Synopsis

```
stap.pass6.end
```

Values

session the systemtap_session variable s

Description

pass6.end fires just before main's return.

Name

probe::stap.cache_clean — Removing file from stap cache

Synopsis

```
stap.cache_clean
```

Values

path the path to the .ko/.c file being removed

Description

Fires just before the call to unlink the module/source file.

Name

probe::stap.cache_add_mod — Adding kernel instrumentation module to cache

Synopsis

```
stap.cache_add_mod
```

Values

<i>dest_path</i>	the path the .ko file is going to (incl filename)
<i>source_path</i>	the path the .ko file is coming from (incl filename)

Description

Fires just before the file is actually moved. Note: if moving fails, `cache_add_src` and `cache_add_nss` will not fire.

Name

probe::stap.cache_add_src — Adding C code translation to cache

Synopsis

```
stap.cache_add_src
```

Values

<i>dest_path</i>	the path the .c file is going to (incl filename)
<i>source_path</i>	the path the .c file is coming from (incl filename)

Description

Fires just before the file is actually moved. Note: if moving the kernel module fails, this probe will not fire.

Name

probe::stap.cache_add_nss — Add NSS (Network Security Services) information to cache

Synopsis

```
stap.cache_add_nss
```

Values

<i>dest_path</i>	the path the .sgn file is coming from (incl filename)
<i>source_path</i>	the path the .sgn file is coming from (incl filename)

Description

Fires just before the file is actually moved. Note: stap must compiled with NSS support; if moving the kernel module fails, this probe will not fire.

Name

probe::stap.cache_get — Found item in stap cache

Synopsis

```
stap.cache_get
```

Values

<i>source_path</i>	the path of the .c source file
<i>module_path</i>	the path of the .ko kernel module file

Description

Fires just before the return of `get_from_cache`, when the cache grab is successful.

Name

probe::stap.system — Starting a command from stap

Synopsis

```
stap.system
```

Values

command the command string to be run by posix_spawn (as sh -c <str>)

Description

Fires at the entry of the stap_system command.

Name

probe::stap.system.spawn — stap spawned new process

Synopsis

```
stap.system.spawn
```

Values

ret the return value from `posix_spawn`

pid the pid of the spawned process

Description

Fires just after the call to `posix_spawn`.

Name

probe::stap.system.return — Finished a command from stap

Synopsis

```
stap.system.return
```

Values

ret a return code associated with running waitpid on the spawned process; a non-zero value indicates error

Description

Fires just before the return of the `stap_system` function, after `waitpid`.

Name

probe::staprun.insert_module — Inserting SystemTap instrumentation module

Synopsis

```
staprun.insert_module
```

Values

path the full path to the .ko kernel module about to be inserted

Description

Fires just before the call to insert the module.

Name

probe::staprun.remove_module — Removing SystemTap instrumentation module

Synopsis

```
staprun.remove_module
```

Values

name the stap module name to be removed (without the .ko extension)

Description

Fires just before the call to remove the module.

Name

probe::staprun.send_control_message — Sending a control message

Synopsis

```
staprun.send_control_message
```

Values

len the length (in bytes) of the data blob

data a ptr to a binary blob of data sent as the control message

type type of message being send; defined in runtime/transport/transport_msgs.h

Description

Fires at the beginning of the send_request function.

Name

probe::stapio.receive_control_message — Recieved a control message

Synopsis

```
stapio.receive_control_message
```

Values

len the length (in bytes) of the data blob

data a ptr to a binary blob of data sent as the control message

type type of message being send; defined in runtime/transport/transport_msgs.h

Description

Fires just after a message was receieved and before it's processed.