

SystemTap Tapset Reference Manual

SystemTap

SystemTap Tapset Reference Manual

by SystemTap

Copyright © 2008-2013 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., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

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

Table of Contents

1. Introduction	1
2. Context Functions	2
function::addr	3
function::asmlinkage	4
function::backtrace	5
function::caller	6
function::caller_addr	7
function::callers	8
function::cmdline_arg	9
function::cmdline_args	10
function::cmdline_str	11
function::cpu	12
function::cpuid	13
function::egid	14
function::env_var	15
function::euid	16
function::execname	17
function::fastcall	18
function::gid	19
function::int_arg	20
function::is_myproc	21
function::is_return	22
function::long_arg	23
function::longlong_arg	24
function::modname	25
function::module_name	26
function::ns_egid	27
function::ns_euid	28
function::ns_gid	29
function::ns_pgrp	30
function::ns_pid	31
function::ns_ppid	32
function::ns_sid	33
function::ns_tid	34
function::ns_uid	35
function::pexecname	36
function::pgrp	37
function::pid	38
function::pid2execname	39
function::pid2task	40
function::pn	41
function::pnlabel	42
function::pointer_arg	43
function::pp	44
function::ppfunc	45
function::ppid	46
function::print_backtrace	47
function::print_regs	48
function::print_stack	49
function::print_syms	50
function::print_ubacktrace	51
function::print_ubacktrace_brief	52
function::print_ustack	53
function::print_usyms	54
function::probe_type	55

function::probefunc	56
function::probemod	57
function::pstrace	58
function::register	59
function::registers_valid	60
function::regparm	61
function::remote_id	62
function::remote_uri	63
function::s32_arg	64
function::s64_arg	65
function::sid	66
function::sprintf_backtrace	67
function::sprintf_stack	68
function::sprintf_syms	69
function::sprintf_ubacktrace	70
function::sprintf_ustack	71
function::sprintf_usyms	72
function::stack	73
function::stack_size	74
function::stack_unused	75
function::stack_used	76
function::stp_pid	77
function::symdata	78
function::symfile	79
function::symfileline	80
function::symline	81
function::symname	82
function::target	83
function::task_ancestry	84
function::task_backtrace	85
function::task_cpu	86
function::task_current	87
function::task_egid	88
function::task_euid	89
function::task_execname	90
function::task_gid	91
function::task_max_file_handles	92
function::task_nice	93
function::task_ns_egid	94
function::task_ns_euid	95
function::task_ns_gid	96
function::task_ns_pid	97
function::task_ns_tid	98
function::task_ns_uid	99
function::task_open_file_handles	100
function::task_parent	101
function::task_pid	102
function::task_prio	103
function::task_state	104
function::task_tid	105
function::task_uid	106
function::tid	107
function::u32_arg	108
function::u64_arg	109
function::u_register	110
function::uaddr	111
function::ubacktrace	112
function::ucallers	113

function::uid	114
function::uint_arg	115
function::ulong_arg	116
function::ulonglong_arg	117
function::umodname	118
function::user_mode	119
function::ustack	120
function::usymdata	121
function::usymfile	122
function::usymfileline	123
function::usymline	124
function::usymname	125
3. Timestamp Functions	126
function::HZ	127
function::cpu_clock_ms	128
function::cpu_clock_ns	129
function::cpu_clock_s	130
function::cpu_clock_us	131
function::delete_stopwatch	132
function::get_cycles	133
function::gettimeofday_ms	134
function::gettimeofday_ns	135
function::gettimeofday_s	136
function::gettimeofday_us	137
function::jiffies	138
function::local_clock_ms	139
function::local_clock_ns	140
function::local_clock_s	141
function::local_clock_us	142
function::read_stopwatch_ms	143
function::read_stopwatch_ns	144
function::read_stopwatch_s	145
function::read_stopwatch_us	146
function::start_stopwatch	147
function::stop_stopwatch	148
4. Time utility functions	149
function::ctime	150
function::tz_ctime	151
function::tz_gmtoff	152
function::tz_name	153
5. Shell command functions	154
function::system	155
6. Memory Tapset	156
function::addr_to_node	157
function::bytes_to_string	158
function::mem_page_size	159
function::pages_to_string	160
function::proc_mem_data	161
function::proc_mem_data_pid	162
function::proc_mem_rss	163
function::proc_mem_rss_pid	164
function::proc_mem_shr	165
function::proc_mem_shr_pid	166
function::proc_mem_size	167
function::proc_mem_size_pid	168
function::proc_mem_string	169
function::proc_mem_string_pid	170
function::proc_mem_txt	171

function::proc_mem_txt_pid	172
function::vm_fault_contains	173
probe::vm.brk	174
probe::vm.kfree	175
probe::vm.kmalloc	176
probe::vm.kmalloc_node	177
probe::vm.kmem_cache_alloc	178
probe::vm.kmem_cache_alloc_node	179
probe::vm.kmem_cache_free	180
probe::vm.mmap	181
probe::vm.munmap	182
probe::vm.oom_kill	183
probe::vm.pagefault	184
probe::vm.pagefault.return	185
probe::vm.write_shared	186
probe::vm.write_shared_copy	187
7. Task Time Tapset	188
function::cputime_to_msecs	189
function::cputime_to_string	190
function::cputime_to_usecs	191
function::msecs_to_string	192
function::nsecs_to_string	193
function::task_start_time	194
function::task_stime	195
function::task_stime_tid	196
function::task_time_string	197
function::task_time_string_tid	198
function::task_untime	199
function::task_untime_tid	200
function::usecs_to_string	201
8. Scheduler Tapset	202
probe::scheduler.balance	203
probe::scheduler.cpu_off	204
probe::scheduler.cpu_on	205
probe::scheduler.ctxswitch	206
probe::scheduler.kthread_stop	207
probe::scheduler.kthread_stop.return	208
probe::scheduler.migrate	209
probe::scheduler.process_exit	210
probe::scheduler.process_fork	211
probe::scheduler.process_free	212
probe::scheduler.process_wait	213
probe::scheduler.signal_send	214
probe::scheduler.tick	215
probe::scheduler.wait_task	216
probe::scheduler.wakeup	217
probe::scheduler.wakeup_new	218
9. IO Scheduler and block IO Tapset	219
probe::ioblock.end	220
probe::ioblock.request	221
probe::ioblock_trace.bounce	222
probe::ioblock_trace.end	223
probe::ioblock_trace.request	224
probe::ioscheduler.elv_add_request	225
probe::ioscheduler.elv_add_request.kp	226
probe::ioscheduler.elv_add_request.tp	227
probe::ioscheduler.elv_completed_request	228
probe::ioscheduler.elv_next_request	229

probe::ioscheduler.elv_next_request.return	230
probe::ioscheduler_trace.elv_abort_request	231
probe::ioscheduler_trace.elv_completed_request	232
probe::ioscheduler_trace.elv_issue_request	233
probe::ioscheduler_trace.elv_requeue_request	234
probe::ioscheduler_trace.plugin	235
probe::ioscheduler_trace.unplug_io	236
probe::ioscheduler_trace.unplug_timer	237
10. SCSI Tapset	238
probe::scsi.iocompleted	239
probe::scsi.iodispatching	240
probe::scsi.iodone	241
probe::scsi.ioentry	242
probe::scsi.ioexecute	243
probe::scsi.set_state	244
11. TTY Tapset	245
probe::tty.init	246
probe::tty.ioctl	247
probe::tty.open	248
probe::tty.poll	249
probe::tty.read	250
probe::tty.receive	251
probe::tty.register	252
probe::tty.release	253
probe::tty.resize	254
probe::tty.unregister	255
probe::tty.write	256
12. Interrupt Request (IRQ) Tapset	257
probe::irq_handler.entry	258
probe::irq_handler.exit	259
probe::softirq.entry	260
probe::softirq.exit	261
probe::workqueue.create	262
probe::workqueue.destroy	263
probe::workqueue.execute	264
probe::workqueue.insert	265
13. Networking Tapset	266
function::format_ipaddr	267
function::htonl	268
function::htonll	269
function::htons	270
function::ip_ntop	271
function::ntohl	272
function::ntohll	273
function::ntohs	274
probe::netdev.change_mac	275
probe::netdev.change_mtu	276
probe::netdev.change_rx_flag	277
probe::netdev.close	278
probe::netdev.get_stats	279
probe::netdev.hard_transmit	280
probe::netdev.ioctl	281
probe::netdev.open	282
probe::netdev.receive	283
probe::netdev.register	284
probe::netdev.rx	285
probe::netdev.set_promiscuity	286
probe::netdev.transmit	287

probe::netdev.unregister	288
probe::netfilter.arp.forward	289
probe::netfilter.arp.in	290
probe::netfilter.arp.out	291
probe::netfilter.bridge.forward	292
probe::netfilter.bridge.local_in	294
probe::netfilter.bridge.local_out	296
probe::netfilter.bridge.post_routing	298
probe::netfilter.bridge.pre_routing	300
probe::netfilter.ip.forward	302
probe::netfilter.ip.local_in	304
probe::netfilter.ip.local_out	306
probe::netfilter.ip.post_routing	308
probe::netfilter.ip.pre_routing	310
probe::sunrpc.clnt.bind_new_program	312
probe::sunrpc.clnt.call_async	313
probe::sunrpc.clnt.call_sync	314
probe::sunrpc.clnt.clone_client	315
probe::sunrpc.clnt.create_client	316
probe::sunrpc.clnt.restart_call	317
probe::sunrpc.clnt.shutdown_client	318
probe::sunrpc.sched.delay	319
probe::sunrpc.sched.execute	320
probe::sunrpc.sched.new_task	321
probe::sunrpc.sched.release_task	322
probe::sunrpc.svc.create	323
probe::sunrpc.svc.destroy	324
probe::sunrpc.svc.drop	325
probe::sunrpc.svc.process	326
probe::sunrpc.svc.recv	327
probe::sunrpc.svc.register	328
probe::sunrpc.svc.send	329
probe::tcp.disconnect	330
probe::tcp.disconnect.return	331
probe::tcp.receive	332
probe::tcp.recvmsg	333
probe::tcp.recvmsg.return	334
probe::tcp.sendmsg	335
probe::tcp.sendmsg.return	336
probe::tcp.setsockopt	337
probe::tcp.setsockopt.return	338
probe::udp.disconnect	339
probe::udp.disconnect.return	340
probe::udp.recvmsg	341
probe::udp.recvmsg.return	342
probe::udp.sendmsg	343
probe::udp.sendmsg.return	344
14. Socket Tapset	345
function::inet_get_ip_source	346
function::inet_get_local_port	347
function::sock_fam_num2str	348
function::sock_fam_str2num	349
function::sock_prot_num2str	350
function::sock_prot_str2num	351
function::sock_state_num2str	352
function::sock_state_str2num	353
probe::socket.aio_read	354
probe::socket.aio_read.return	355

probe::socket.aio_write	356
probe::socket.aio_write.return	357
probe::socket.close	358
probe::socket.close.return	359
probe::socket.create	360
probe::socket.create.return	361
probe::socket.read_iter	362
probe::socket.read_iter.return	363
probe::socket.readv	364
probe::socket.readv.return	365
probe::socket.receive	366
probe::socket.recvmsg	367
probe::socket.recvmsg.return	368
probe::socket.send	369
probe::socket.sendmsg	370
probe::socket.sendmsg.return	371
probe::socket.write_iter	372
probe::socket.write_iter.return	373
probe::socket.writev	374
probe::socket.writev.return	375
15. SNMP Information Tapset	376
function::ipmib_filter_key	377
function::ipmib_get_proto	378
function::ipmib_local_addr	379
function::ipmib_remote_addr	380
function::ipmib_tcp_local_port	381
function::ipmib_tcp_remote_port	382
function::linuxmib_filter_key	383
function::tcpmib_filter_key	384
function::tcpmib_get_state	385
function::tcpmib_local_addr	386
function::tcpmib_local_port	387
function::tcpmib_remote_addr	388
function::tcpmib_remote_port	389
probe::ipmib.ForwDatagrams	390
probe::ipmib.FragFails	391
probe::ipmib.FragOKs	392
probe::ipmib.InAddrErrors	393
probe::ipmib.InDiscards	394
probe::ipmib.InNoRoutes	395
probe::ipmib.InReceives	396
probe::ipmib.InUnknownProtos	397
probe::ipmib.OutRequests	398
probe::ipmib.ReasmReqds	399
probe::ipmib.ReasmTimeout	400
probe::linuxmib.DelayedACKs	401
probe::linuxmib.ListenDrops	402
probe::linuxmib.ListenOverflows	403
probe::linuxmib.TCPMemoryPressures	404
probe::tcpmib.ActiveOpens	405
probe::tcpmib.AttemptFails	406
probe::tcpmib.CurrEstab	407
probe::tcpmib.EstabResets	408
probe::tcpmib.InSegs	409
probe::tcpmib.OutRsts	410
probe::tcpmib.OutSegs	411
probe::tcpmib.PassiveOpens	412
probe::tcpmib.RetransSegs	413

16. Kernel Process Tapset	414
function::get_loadavg_index	415
function::sprint_loadavg	416
function::target_set_pid	417
function::target_set_report	418
probe::kprocess.create	419
probe::kprocess.exec	420
probe::kprocess.exec_complete	421
probe::kprocess.exit	422
probe::kprocess.release	423
probe::kprocess.start	424
17. Signal Tapset	425
function::get_sa_flags	426
function::get_sa_handler	427
function::is_sig_blocked	428
function::sa_flags_str	429
function::sa_handler_str	430
function::signal_str	431
function::sigset_mask_str	432
probe::signal.check_ignored	433
probe::signal.check_ignored.return	434
probe::signal.checkperm	435
probe::signal.checkperm.return	436
probe::signal.do_action	437
probe::signal.do_action.return	438
probe::signal.flush	439
probe::signal.force_segv	440
probe::signal.force_segv.return	441
probe::signal.handle	442
probe::signal.handle.return	443
probe::signal.pending	444
probe::signal.pending.return	445
probe::signal.procmask	446
probe::signal.procmask.return	447
probe::signal.send	448
probe::signal.send.return	449
probe::signal.send_sig_queue	450
probe::signal.send_sig_queue.return	451
probe::signal.sys_tkill	452
probe::signal.sys_tkill.return	453
probe::signal.sys_tkill	454
probe::signal.syskill	455
probe::signal.syskill.return	456
probe::signal.systkill.return	457
probe::signal.wakeup	458
18. Errno Tapset	459
function::errno_str	460
function::return_str	461
function::returnstr	462
function::returnval	463
19. RLIMIT Tapset	464
function::rlimit_from_str	465
20. Device Tapset	466
function::MAJOR	467
function::MINOR	468
function::MKDEV	469
function::usrdev2kerndev	470
21. Directory-entry (dentry) Tapset	471

function::d_name	472
function::d_path	473
function::inode_name	474
function::inode_path	475
function::real_mount	476
function::reverse_path_walk	477
function::task_dentry_path	478
22. Logging Tapset	479
function::error	480
function::exit	481
function::ftrace	482
function::log	483
function::printk	484
function::warn	485
23. Queue Statistics Tapset	486
function::qs_done	487
function::qs_run	488
function::qs_wait	489
function::qsq_blocked	490
function::qsq_print	491
function::qsq_service_time	492
function::qsq_start	493
function::qsq_throughput	494
function::qsq_utilization	495
function::qsq_wait_queue_length	496
function::qsq_wait_time	497
24. Random functions Tapset	498
function::randint	499
25. String and data retrieving functions Tapset	500
function::atomic_long_read	501
function::atomic_read	502
function::kernel_char	503
function::kernel_int	504
function::kernel_long	505
function::kernel_pointer	506
function::kernel_short	507
function::kernel_string	508
function::kernel_string2	509
function::kernel_string2_utf16	510
function::kernel_string2_utf32	511
function::kernel_string_n	512
function::kernel_string_quoted	513
function::kernel_string_utf16	514
function::kernel_string_utf32	515
function::user_char	516
function::user_char_warn	517
function::user_int	518
function::user_int16	519
function::user_int32	520
function::user_int64	521
function::user_int8	522
function::user_int_warn	523
function::user_long	524
function::user_long_warn	525
function::user_short	526
function::user_short_warn	527
function::user_string	528
function::user_string2	529

function::user_string2_n_warn	530
function::user_string2_utf16	531
function::user_string2_utf32	532
function::user_string2_warn	533
function::user_string_n	534
function::user_string_n2	535
function::user_string_n2_quoted	536
function::user_string_n_quoted	537
function::user_string_n_warn	538
function::user_string_quoted	539
function::user_string_utf16	540
function::user_string_utf32	541
function::user_string_warn	542
function::user_uint16	543
function::user_uint32	544
function::user_uint64	545
function::user_uint8	546
function::user_ulong	547
function::user_ulong_warn	548
function::user_ushort	549
function::user_ushort_warn	550
26. String and data writing functions Tapset	551
function::set_kernel_char	552
function::set_kernel_int	553
function::set_kernel_long	554
function::set_kernel_pointer	555
function::set_kernel_short	556
function::set_kernel_string	557
function::set_kernel_string_n	558
27. Guru tapsets	559
function::mdelay	560
function::panic	561
function::raise	562
function::udelay	563
28. A collection of standard string functions	564
function::isdigit	565
function::isinstr	566
function::str_replace	567
function::stringat	568
function::strlen	569
function::strtol	570
function::substr	571
function::text_str	572
function::text_strn	573
function::tokenize	574
29. Utility functions for using ansi control chars in logs	575
function::ansi_clear_screen	576
function::ansi_cursor_hide	577
function::ansi_cursor_move	578
function::ansi_cursor_restore	579
function::ansi_cursor_save	580
function::ansi_cursor_show	581
function::ansi_new_line	582
function::ansi_reset_color	583
function::ansi_set_color	584
function::ansi_set_color2	585
function::ansi_set_color3	586
function::indent	587

function::indent_depth	588
function::thread_indent	589
function::thread_indent_depth	590
30. SystemTap Translator Tapset	591
probe::stap.cache_add_mod	592
probe::stap.cache_add_nss	593
probe::stap.cache_add_src	594
probe::stap.cache_clean	595
probe::stap.cache_get	596
probe::stap.pass0	597
probe::stap.pass0.end	598
probe::stap.pass1.end	599
probe::stap.pass1a	600
probe::stap.pass1b	601
probe::stap.pass2	602
probe::stap.pass2.end	603
probe::stap.pass3	604
probe::stap.pass3.end	605
probe::stap.pass4	606
probe::stap.pass4.end	607
probe::stap.pass5	608
probe::stap.pass5.end	609
probe::stap.pass6	610
probe::stap.pass6.end	611
probe::stap.system	612
probe::stap.system.return	613
probe::stap.system.spawn	614
probe::stapio.receive_control_message	615
probe::staprun.insert_module	616
probe::staprun.remove_module	617
probe::staprun.send_control_message	618
31. Network File Storage Tapsets	619
function::nfsderror	620
probe::nfs.aop.readpage	621
probe::nfs.aop.readpages	622
probe::nfs.aop.release_page	623
probe::nfs.aop.set_page_dirty	624
probe::nfs.aop.write_begin	625
probe::nfs.aop.write_end	626
probe::nfs.aop.writepage	627
probe::nfs.aop.writepages	628
probe::nfs.fop.aio_read	629
probe::nfs.fop.aio_write	630
probe::nfs.fop.check_flags	631
probe::nfs.fop.flush	632
probe::nfs.fop.fsync	633
probe::nfs.fop.llseek	634
probe::nfs.fop.lock	635
probe::nfs.fop.mmap	636
probe::nfs.fop.open	637
probe::nfs.fop.read	638
probe::nfs.fop.read_iter	639
probe::nfs.fop.release	640
probe::nfs.fop.sendfile	641
probe::nfs.fop.write	642
probe::nfs.fop.write_iter	643
probe::nfs.proc.commit	644
probe::nfs.proc.commit_done	645

probe::nfs.proc.commit_setup	646
probe::nfs.proc.create	647
probe::nfs.proc.handle_exception	648
probe::nfs.proc.lookup	649
probe::nfs.proc.open	650
probe::nfs.proc.read	651
probe::nfs.proc.read_done	652
probe::nfs.proc.read_setup	653
probe::nfs.proc.release	654
probe::nfs.proc.remove	655
probe::nfs.proc.rename	656
probe::nfs.proc.rename_done	657
probe::nfs.proc.rename_setup	658
probe::nfs.proc.write	659
probe::nfs.proc.write_done	660
probe::nfs.proc.write_setup	661
probe::nfsd.close	662
probe::nfsd.commit	663
probe::nfsd.create	664
probe::nfsd.createv3	665
probe::nfsd.dispatch	666
probe::nfsd.lookup	667
probe::nfsd.open	668
probe::nfsd.proc.commit	669
probe::nfsd.proc.create	670
probe::nfsd.proc.lookup	671
probe::nfsd.proc.read	672
probe::nfsd.proc.remove	673
probe::nfsd.proc.rename	674
probe::nfsd.proc.write	675
probe::nfsd.read	676
probe::nfsd.rename	677
probe::nfsd.unlink	678
probe::nfsd.write	679
32. Speculation	680
function::commit	681
function::discard	682
function::speculate	683
function::speculation	684
33. JSON Tapset	685
function::json_add_array	686
function::json_add_array_numeric_metric	687
function::json_add_array_string_metric	688
function::json_add_numeric_metric	689
function::json_add_string_metric	690
function::json_set_prefix	691
macro::json_output_array_numeric_value	692
macro::json_output_array_string_value	693
macro::json_output_data_end	694
macro::json_output_data_start	695
macro::json_output_numeric_value	696
macro::json_output_string_value	697
probe::json_data	698

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.

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::addr — Address of the current probe point.

Synopsis

```
addr:long()
```

Arguments

None

Description

Returns the instruction pointer from the current probe's register state. Not all probe types have registers though, in which case zero is returned. The returned address is suitable for use with functions like `symname` and `symdata`.

Name

function::asmlinkage — Mark function as declared asmlinkage

Synopsis

```
asmlinkage()
```

Arguments

None

Description

Call this function before accessing arguments using the *_arg functions if the probed kernel function was declared asmlinkage in the source.

Name

function::backtrace — Hex backtrace of current kernel stack

Synopsis

```
backtrace:string()
```

Arguments

None

Description

This function returns a string of hex addresses that are a backtrace of the kernel stack. Output may be truncated as per maximum string length (MAXSTRINGLEN). See `ubacktrace` for user-space backtrace.

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)`

Name

function::caller_addr — Return caller address

Synopsis

```
caller_addr:long()
```

Arguments

None

Description

This function returns the address of the calling function.

Name

function::callers — Return first *n* elements of kernel stack backtrace

Synopsis

```
callers:string(n:long)
```

Arguments

n — number of levels to descend in the stack (not counting the top level). If *n* is -1, print the entire stack.

Description

This function returns a string of the first *n* hex addresses from the backtrace of the kernel stack. Output may be truncated as per maximum string length (MAXSTRINGLEN).

Name

function::cmdline_arg — Fetch a command line argument

Synopsis

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

Arguments

n Argument to get (zero is the program 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_args` — Fetch command line arguments from current process

Synopsis

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

Arguments

n First argument to get (zero is normally the program itself)

m Last argument to get (or minus one for all arguments after *n*)

delim String to use to separate 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_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::cpu — Returns the current cpu number

Synopsis

```
cpu:long()
```

Arguments

None

Description

This function returns the current cpu number.

Name

function::cpuid — Returns the current cpu number

Synopsis

```
cpuid:long()
```

Arguments

None

Description

This function returns the current cpu number. Deprecated in SystemTap 1.4 and removed in SystemTap 1.5.

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::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::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::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::fastcall — Mark function as declared fastcall

Synopsis

```
fastcall()
```

Arguments

None

Description

Call this function before accessing arguments using the *_arg functions if the probed kernel function was declared fastcall in the source.

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::int_arg` — Return function argument as signed int

Synopsis

```
int_arg:long(n:long)
```

Arguments

n index of argument to return

Description

Return the value of argument *n* as a signed int (i.e., a 32-bit integer sign-extended to 64 bits).

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::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::long_arg — Return function argument as signed long

Synopsis

```
long_arg:long(n:long)
```

Arguments

n index of argument to return

Description

Return the value of argument *n* as a signed long. On architectures where a long is 32 bits, the value is sign-extended to 64 bits.

Name

`function::longlong_arg` — Return function argument as 64-bit value

Synopsis

```
longlong_arg:long(n:long)
```

Arguments

n index of argument to return

Description

Return the value of argument *n* as a 64-bit value.

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 raise an error. If the address was not in a kernel module, but in the kernel itself, then the string “kernel” will be returned.

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::ns_egid — Returns the effective gid of a target process as seen in a user namespace

Synopsis

```
ns_egid:long()
```

Arguments

None

Description

This function returns the effective gid of a target process as seen in the target user namespace if provided, or the stap process namespace

Name

function::ns_euid — Returns the effective user ID of a target process as seen in a user namespace

Synopsis

```
ns_euid:long()
```

Arguments

None

Description

This function returns the effective user ID of the target process as seen in the target user namespace if provided, or the stap process namespace.

Name

function::ns_gid — Returns the group ID of a target process as seen in a user namespace

Synopsis

```
ns_gid:long()
```

Arguments

None

Description

This function returns the group ID of a target process as seen in the target user namespace if provided, or the stap process namespace.

Name

function::ns_pgrp — Returns the process group ID of the current process as seen in a pid namespace

Synopsis

```
ns_pgrp:long( )
```

Arguments

None

Description

This function returns the process group ID of the current process as seen in the target pid namespace if provided, or the stap process namespace.

Name

function::ns_pid — Returns the ID of a target process as seen in a pid namespace

Synopsis

```
ns_pid:long()
```

Arguments

None

Description

This function returns the ID of a target process as seen in the target pid namespace.

Name

function::ns_ppid — Returns the process ID of a target process's parent process as seen in a pid namespace

Synopsis

```
ns_ppid:long()
```

Arguments

None

Description

This function return the process ID of the target proccess's parent process as seen in the target pid namespace if provided, or the stap process namespace.

Name

function::ns_sid — Returns the session ID of the current process as seen in a pid namespace

Synopsis

```
ns_sid:long()
```

Arguments

None

Description

The namespace-aware session ID of a process is the process group ID of the session leader as seen in the target pid namespace if provided, or the stap process namespace. Session ID is stored in the `signal_struct` since Kernel 2.6.0.

Name

function::ns_tid — Returns the thread ID of a target process as seen in a pid namespace

Synopsis

```
ns_tid:long()
```

Arguments

None

Description

This function returns the thread ID of a target process as seen in the target pid namespace if provided, or the stap process namespace.

Name

function::ns_uid — Returns the user ID of a target process as seen in a user namespace

Synopsis

```
ns_uid:long()
```

Arguments

None

Description

This function returns the user ID of the target process as seen in the target user namespace if provided, or the stap process namespace.

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::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::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::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::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::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.

Name

function::pnlabel — Returns the label name parsed from the probe name

Synopsis

```
pnlabel:string()
```

Arguments

None

Description

This returns the label name as parsed from the script-level probe point. This function will only work if called directly from the body of a '.label' probe point (i.e. no aliases).

Context

The current probe point.

Name

function::pointer_arg — Return function argument as pointer value

Synopsis

```
pointer_arg:long(n:long)
```

Arguments

n index of argument to return

Description

Return the unsigned value of argument *n*, same as `ulong_arg`. Can be used with any type of pointer.

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::ppfunc` — Returns the function name parsed from `pp`

Synopsis

```
ppfunc:string()
```

Arguments

None

Description

This returns the function name from the current `pp`. Not all `pp` have functions in them, in which case `""` is returned.

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::print_backtrace — Print kernel 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. See `print_ubacktrace` for user-space backtrace. The function does not return a value.

Name

function::print_regs — Print a register dump

Synopsis

```
print_regs()
```

Arguments

None

Description

This function prints a register dump. Does nothing if no registers are available for the probe point.

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.

NOTE

it is recommended to use `print_syms` instead of this function.

Name

function::print_syms — Print out kernel stack from string

Synopsis

```
print_syms(callers:string)
```

Arguments

callers String with list of hexadecimal (kernel) addresses

Description

This function performs a symbolic lookup of the addresses in the given string, which are assumed to be the result of prior calls to `stack`, `callers`, and similar functions.

Prints one line per address, including the address, the name of the function containing the address, and an estimate of its position within that function, as obtained by `symdata`. Returns nothing.

Name

function::print_ubacktrace — Print stack back trace for current user-space task.

Synopsis

```
print_ubacktrace()
```

Arguments

None

Description

Equivalent to `print_ustack(ubacktrace)`, except that deeper stack nesting may be supported. Returns nothing. See `print_backtrace` for kernel 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::print_ubacktrace_brief — Print stack back trace for current user-space task.

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

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.

NOTE

it is recommended to use `print_usyms` instead of this function.

Name

function::print_usyms — Print out user stack from string

Synopsis

```
print_usyms(callers:string)
```

Arguments

callers String with list of hexadecimal (user) addresses

Description

This function performs a symbolic lookup of the addresses in the given string, which are assumed to be the result of prior calls to `ustack`, `ucallers`, and similar functions.

Prints one line per address, including the address, the name of the function containing the address, and an estimate of its position within that function, as obtained by `usymdata`. Returns nothing.

Name

function::probe_type — The low level probe handler type of the current probe.

Synopsis

```
probe_type:string()
```

Arguments

None

Description

Returns a short string describing the low level probe handler type for the current probe point. This is for informational purposes only. Depending on the low level probe handler different context functions can or cannot provide information about the current event (for example some probe handlers only trigger in user space and have no associated kernel context). High-level probes might map to the same or different low-level probes (depending on systemtap version and/or kernel used).

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 based on the current address, as computed by `symname(addr)` or `usymname(uaddr)` depending on probe context (whether the probe is a user probe or a kernel probe).

Please note

this function's behaviour differs between SystemTap 2.0 and earlier versions. Prior to 2.0, `probefunc` obtained the function name from the probe point string as returned by `pp`, and used the current address as a fallback.

Consider using `ppfunc` instead.

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::pstrace — Chain of processes and pids back to init(1)

Synopsis

```
pstrace:string(task:long)
```

Arguments

task Pointer to task struct of process

Description

This function returns a string listing execname and pid for each process starting from *task* back to the process ancestor that init(1) spawned.

Name

function::register — Return the signed value of the named CPU register

Synopsis

```
register:long(name:string)
```

Arguments

name Name of the register to return

Description

Return the value of the named CPU register, as it was saved when the current probe point was hit. If the register is 32 bits, it is sign-extended to 64 bits.

For the i386 architecture, the following names are recognized. (name1/name2 indicates that name1 and name2 are alternative names for the same register.) eax/ax, ebp/bp, ebx/bx, ecx/cx, edi/di, edx/dx, eflags/flags, eip/ip, esi/si, esp/sp, orig_eax/orig_ax, xcs/cs, xds/ds, xes/es, xfs/fs, xss/ss.

For the x86_64 architecture, the following names are recognized: 64-bit registers: r8, r9, r10, r11, r12, r13, r14, r15, rax/ax, rbp/bp, rbx/bx, rcx/cx, rdi/di, rdx/dx, rip/ip, rsi/si, rsp/sp; 32-bit registers: eax, ebp, ebx, ecx, edx, edi, edx, eip, esi, esp, flags/eflags, orig_eax; segment registers: xcs/cs, xss/ss.

For powerpc, the following names are recognized: r0, r1, ... r31, nip, msr, orig_gpr3, ctr, link, xer, ccr, softe, trap, dar, dsisr, result.

For s390x, the following names are recognized: r0, r1, ... r15, args, psw.mask, psw.addr, orig_gpr2, ilc, trap.

For AArch64, the following names are recognized: x0, x1, ... x30, fp, lr, sp, pc, and orig_x0.

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::regparm — Specify regparm value used to compile function

Synopsis

```
regparm(n:long)
```

Arguments

n original regparm value

Description

Call this function with argument *n* before accessing function arguments using the `*_arg` function is the function was build with the `gcc -mregparm=n` option.

(The i386 kernel is built with `\-mregparm=3`, so `systemtap` considers `regparm(3)` the default for kernel functions on that architecture.) Only valid on i386 and x86_64 (when probing 32bit applications). Produces an error on other architectures.

Name

function::remote_id — The index of this instance in a remote execution.

Synopsis

```
remote_id:long()
```

Arguments

None

Description

This function returns a number 0..N, which is the unique index of this particular script execution from a swarm of “stap --remote A --remote B ...” runs, and is the same number “stap --remote-prefix” would print. The function returns -1 if the script was not launched with “stap --remote”, or if the remote staprun/stapsh are older than version 1.7.

Name

`function::remote_uri` — The name of this instance in a remote execution.

Synopsis

```
remote_uri:string()
```

Arguments

None

Description

This function returns the remote host used to invoke this particular script execution from a swarm of “`stap --remote`” runs. It may not be unique among the swarm. The function returns an empty string if the script was not launched with “`stap --remote`”.

Name

function::s32_arg — Return function argument as signed 32-bit value

Synopsis

```
s32_arg:long(n:long)
```

Arguments

n index of argument to return

Description

Return the signed 32-bit value of argument *n*, same as `int_arg`.

Name

function::s64_arg — Return function argument as signed 64-bit value

Synopsis

```
s64_arg:long(n:long)
```

Arguments

n index of argument to return

Description

Return the signed 64-bit value of argument *n*, same as `longlong_arg`.

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::sprint_backtrace — Return stack back trace as string

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::sprint_stack` — Return stack for kernel addresses from string

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`.

NOTE

it is recommended to use `sprint_syms` instead of this function.

Name

`function::sprint_syms` — Return stack for kernel addresses from string

Synopsis

```
sprint_syms(callers:string)
```

Arguments

callers String with list of hexadecimal (kernel) addresses

Description

Perform a symbolic lookup of the addresses in the given string, which are assumed to be the result of a prior calls to `stack`, `callers`, and similar functions.

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), as obtained from `syndata`. 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_syms`.

Name

function::sprint_ubacktrace — Return stack back trace for current user-space task as string.

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 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_ustack` — Return stack for the current task from string.

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`.

NOTE

it is recommended to use `sprint_usyms` instead of this function.

Name

`function::sprint_usyms` — Return stack for user addresses from string

Synopsis

```
sprint_usyms(callers:string)
```

Arguments

callers String with list of hexadecimal (user) addresses

Description

Perform a symbolic lookup of the addresses in the given string, which are assumed to be the result of a prior calls to `ustack`, `ucallers`, and similar functions.

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), as obtained from `usymdata`. 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_usyms`.

Name

`function::stack` — Return address at given depth of kernel stack backtrace

Synopsis

```
stack:long(n:long)
```

Arguments

n number of levels to descend in the stack.

Description

Performs a simple (kernel) backtrace, and returns the element at the specified position. The results of the backtrace itself are cached, so that the backtrace computation is performed at most once no matter how many times `stack` is called, or in what order.

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_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::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::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::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::symfile` — Return the file name of a given address.

Synopsis

```
symfile:string(addr:long)
```

Arguments

addr The address to translate.

Description

Returns the file name of the given address, if known. If the file name cannot be found, the hex string representation of the address will be returned.

Name

`function::symfileline` — Return the file name and line number of an address.

Synopsis

```
symfileline:string(addr:long)
```

Arguments

addr The address to translate.

Description

Returns the file name and the (approximate) line number of the given address, if known. If the file name or the line number cannot be found, the hex string representation of the address will be returned.

Name

`function::symline` — Return the line number of an address.

Synopsis

```
symline:string(addr:long)
```

Arguments

addr The address to translate.

Description

Returns the (approximate) line number of the given address, if known. If the line number cannot be found, the hex string representation of the address 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::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::task_ancestry — The ancestry of the given task

Synopsis

```
task_ancestry:string(task:long,with_time:long)
```

Arguments

task task_struct pointer

with_time set to 1 to also print the start time of processes (given as a delta from boot time)

Description

Return the ancestry of the given task in the form of “grandparent_process=>parent_process=>process”.

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. Deprecated in SystemTap 1.6.

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_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_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_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_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_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_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::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_ns_egid — The effective group identifier of the task

Synopsis

```
task_ns_egid:long(task:long)
```

Arguments

task task_struct pointer

Description

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

Name

function::task_ns_euid — The effective user identifier of the task

Synopsis

```
task_ns_euid:long(task:long)
```

Arguments

task task_struct pointer

Description

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

Name

function::task_ns_gid — The group identifier of the task as seen in a namespace

Synopsis

```
task_ns_gid:long(task:long)
```

Arguments

task task_struct pointer

Description

This function returns the group id of the given task as seen in in the given user namespace.

Name

function::task_ns_pid — The process identifier of the task

Synopsis

```
task_ns_pid:long(task:long)
```

Arguments

task task_struct pointer

Description

This function returns the process id of the given task based on the specified pid namespace..

Name

`function::task_ns_tid` — The thread identifier of the task as seen in a namespace

Synopsis

```
task_ns_tid:long(task:long)
```

Arguments

task task_struct pointer

Description

This function returns the thread id of the given task as seen in the pid namespace.

Name

function::task_ns_uid — The user identifier of the task

Synopsis

```
task_ns_uid:long(task:long)
```

Arguments

task task_struct pointer

Description

This function returns the user id of 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_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_pid — The process identifier of the task

Synopsis

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

Arguments

task task_struct pointer

Description

This function returns the process 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_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_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_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::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::u32_arg — Return function argument as unsigned 32-bit value

Synopsis

```
u32_arg:long(n:long)
```

Arguments

n index of argument to return

Description

Return the unsigned 32-bit value of argument *n*, same as `uint_arg`.

Name

function::u64_arg — Return function argument as unsigned 64-bit value

Synopsis

```
u64_arg:long(n:long)
```

Arguments

n index of argument to return

Description

Return the unsigned 64-bit value of argument *n*, same as `ulonglong_arg`.

Name

`function::u_register` — Return the unsigned value of the named CPU register

Synopsis

```
u_register:long(name:string)
```

Arguments

name Name of the register to return

Description

Same as `register(name)`, except that if the register is 32 bits wide, it is zero-extended to 64 bits.

Name

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

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 `usymdata`. Often the task will be in the VDSO where it entered the kernel.

Name

function::ubacktrace — Hex backtrace of current user-space task stack.

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. See `backtrace` for kernel traceback.

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::ucallers — Return first n elements of user stack backtrace

Synopsis

```
ucallers:string(n:long)
```

Arguments

n number of levels to descend in the stack (not counting the top level). If *n* is -1, print the entire stack.

Description

This function returns a string of the first *n* hex addresses from the backtrace of the user stack. Output may be truncated as per maximum string length (MAXSTRINGLEN).

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::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::uint_arg` — Return function argument as unsigned int

Synopsis

```
uint_arg:long(n:long)
```

Arguments

n index of argument to return

Description

Return the value of argument *n* as an unsigned int (i.e., a 32-bit integer zero-extended to 64 bits).

Name

`function::ulong_arg` — Return function argument as unsigned long

Synopsis

```
ulong_arg:long(n:long)
```

Arguments

n index of argument to return

Description

Return the value of argument *n* as an unsigned long. On architectures where a long is 32 bits, the value is zero-extended to 64 bits.

Name

function::ulonglong_arg — Return function argument as 64-bit value

Synopsis

```
ulonglong_arg:long(n:long)
```

Arguments

n index of argument to return

Description

Return the value of argument *n* as a 64-bit value. (Same as `longlong_arg`.)

Name

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

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. Reports an error when the address isn't in a (mapped in) module, or the module cannot be found for some reason.

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::ustack` — Return address at given depth of user stack backtrace

Synopsis

```
ustack:long(n:long)
```

Arguments

n number of levels to descend in the stack.

Description

Performs a simple (user space) backtrace, and returns the element at the specified position. The results of the backtrace itself are cached, so that the backtrace computation is performed at most once no matter how many times `ustack` is called, or in what order.

Name

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

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::usymfile` — Return the file name of a given address.

Synopsis

```
usymfile:string(addr:long)
```

Arguments

addr The address to translate.

Description

Returns the file name of the given address, if known. If the file name cannot be found, the hex string representation of the address will be returned.

Name

`function::usymfileline` — Return the file name and line number of an address.

Synopsis

```
usymfileline:string(addr:long)
```

Arguments

addr The address to translate.

Description

Returns the file name and the (approximate) line number of the given address, if known. If the file name or the line number cannot be found, the hex string representation of the address will be returned.

Name

`function::usymline` — Return the line number of an address.

Synopsis

```
usymline:string(addr:long)
```

Arguments

addr The address to translate.

Description

Returns the (approximate) line number of the given address, if known. If the line number cannot be found, the hex string representation of the address will be returned.

Name

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

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*.

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::HZ — Kernel HZ

Synopsis

```
HZ:long()
```

Arguments

None

Description

This function returns the value of the kernel HZ macro, which corresponds to the rate of increase of the jiffies value.

Name

`function::cpu_clock_ms` — Number of milliseconds on the given cpu's clock

Synopsis

```
cpu_clock_ms : long ( cpu : long )
```

Arguments

cpu Which processor's clock to read

Description

This function returns the number of milliseconds on the given cpu's clock. This is always monotonic comparing on the same cpu, but may have some drift between cpus (within about a jiffy).

Name

function::cpu_clock_ns — Number of nanoseconds on the given cpu's clock

Synopsis

```
cpu_clock_ns:long(cpu:long)
```

Arguments

cpu Which processor's clock to read

Description

This function returns the number of nanoseconds on the given cpu's clock. This is always monotonic comparing on the same cpu, but may have some drift between cpus (within about a jiffy).

Name

function::cpu_clock_s — Number of seconds on the given cpu's clock

Synopsis

```
cpu_clock_s:long(cpu:long)
```

Arguments

cpu Which processor's clock to read

Description

This function returns the number of seconds on the given cpu's clock. This is always monotonic comparing on the same cpu, but may have some drift between cpus (within about a jiffy).

Name

`function::cpu_clock_us` — Number of microseconds on the given cpu's clock

Synopsis

```
cpu_clock_us:long(cpu:long)
```

Arguments

cpu Which processor's clock to read

Description

This function returns the number of microseconds on the given cpu's clock. This is always monotonic comparing on the same cpu, but may have some drift between cpus (within about a jiffy).

Name

function::delete_stopwatch — Remove an existing stopwatch

Synopsis

```
delete_stopwatch(name:string)
```

Arguments

name the stopwatch name

Description

Remove stopwatch *name*.

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_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_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_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.

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::jiffies — Kernel jiffies count

Synopsis

```
jiffies:long()
```

Arguments

None

Description

This function returns the value of the kernel jiffies variable. This value is incremented periodically by timer interrupts, and may wrap around a 32-bit or 64-bit boundary. See HZ.

Name

function::local_clock_ms — Number of milliseconds on the local cpu's clock

Synopsis

```
local_clock_ms:long()
```

Arguments

None

Description

This function returns the number of milliseconds on the local cpu's clock. This is always monotonic comparing on the same cpu, but may have some drift between cpus (within about a jiffy).

Name

function::local_clock_ns — Number of nanoseconds on the local cpu's clock

Synopsis

```
local_clock_ns:long()
```

Arguments

None

Description

This function returns the number of nanoseconds on the local cpu's clock. This is always monotonic comparing on the same cpu, but may have some drift between cpus (within about a jiffy).

Name

function::local_clock_s — Number of seconds on the local cpu's clock

Synopsis

```
local_clock_s:long()
```

Arguments

None

Description

This function returns the number of seconds on the local cpu's clock. This is always monotonic comparing on the same cpu, but may have some drift between cpus (within about a jiffy).

Name

function::local_clock_us — Number of microseconds on the local cpu's clock

Synopsis

```
local_clock_us:long()
```

Arguments

None

Description

This function returns the number of microseconds on the local cpu's clock. This is always monotonic comparing on the same cpu, but may have some drift between cpus (within about a jiffy).

Name

`function::read_stopwatch_ms` — Reads the time in milliseconds for a stopwatch

Synopsis

```
read_stopwatch_ms:long(name:string)
```

Arguments

name stopwatch name

Description

Returns time in milliseconds for stopwatch *name*. Creates stopwatch *name* if it does not currently exist.

Name

`function::read_stopwatch_ns` — Reads the time in nanoseconds for a stopwatch

Synopsis

```
read_stopwatch_ns:long(name:string)
```

Arguments

name stopwatch name

Description

Returns time in nanoseconds for stopwatch *name*. Creates stopwatch *name* if it does not currently exist.

Name

`function::read_stopwatch_s` — Reads the time in seconds for a stopwatch

Synopsis

```
read_stopwatch_s:long(name:string)
```

Arguments

name stopwatch name

Description

Returns time in seconds for stopwatch *name*. Creates stopwatch *name* if it does not currently exist.

Name

`function::read_stopwatch_us` — Reads the time in microseconds for a stopwatch

Synopsis

```
read_stopwatch_us:long(name:string)
```

Arguments

name stopwatch name

Description

Returns time in microseconds for stopwatch *name*. Creates stopwatch *name* if it does not currently exist.

Name

function::start_stopwatch — Start a stopwatch

Synopsis

```
start_stopwatch(name:string)
```

Arguments

name the stopwatch name

Description

Start stopwatch *name*. Creates stopwatch *name* if it does not currently exist.

Name

function::stop_stopwatch — Stop a stopwatch

Synopsis

```
stop_stopwatch(name:string)
```

Arguments

name the stopwatch name

Description

Stop stopwatch *name*. Creates stopwatch *name* if it does not currently exist.

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) an error will occur (which can be avoided with try/catch). If the time would be unreasonable far in the future, an error will also occur.

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_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.

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.

Chapter 5. Shell command functions

Utility functions to enqueue shell commands.

Name

function::system — Issue a command to the system

Synopsis

```
system(cmd:string)
```

Arguments

cmd the command to issue to the system

Description

This function runs a command on the system. The command is started in the background some time after the current probe completes. The command is run with the same UID as the user running the `stap` or `staprun` command.

Chapter 6. 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::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

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

Synopsis

```
mem_page_size:long()
```

Arguments

None

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_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::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_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_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”.

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::vm_fault_contains — Test return value for page fault reason

Synopsis

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

Arguments

value the fault_type returned by vm.page_fault.return

test the type of fault to test for (VM_FAULT_OOM or similar)

Name

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

Synopsis

`vm.brk`

Values

length the length of the memory segment

name name of the probe point

address the requested address

Context

The process calling brk.

Name

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

Synopsis

`vm.kfree`

Values

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

Name

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

Synopsis

`vm.kmalloc`

Values

<i>gfp_flags</i>	type of kmemory to allocate
<i>ptr</i>	pointer to the kmemory allocated
<i>gfp_flag_name</i>	type of kmemory to allocate (in String format)
<i>bytes_alloc</i>	allocated Bytes
<i>call_site</i>	address of the kmemory function
<i>bytes_req</i>	requested Bytes
<i>name</i>	name of the probe point
<i>caller_function</i>	name of the caller function

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>gfp_flags</i>	type of kmemory to allocate
<i>bytes_alloc</i>	allocated Bytes
<i>gfp_flag_name</i>	type of kmemory to allocate(in string format)
<i>bytes_req</i>	requested Bytes
<i>call_site</i>	address of the function caling this kmemory function
<i>caller_function</i>	name of the caller function
<i>name</i>	name of the probe point

Name

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

Synopsis

vm.kmem_cache_alloc

Values

<i>bytes_alloc</i>	allocated Bytes
<i>gfp_flag_name</i>	type of kmemory to allocate(in string format)
<i>ptr</i>	pointer to the kmemory allocated
<i>gfp_flags</i>	type of kmemory to allocate
<i>caller_function</i>	name of the caller function.
<i>name</i>	name of the probe point
<i>call_site</i>	address of the function calling this kmemory function.
<i>bytes_req</i>	requested Bytes

Name

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

Synopsis

vm.kmem_cache_alloc_node

Values

<i>name</i>	name of the probe point
<i>caller_function</i>	name of the caller function
<i>call_site</i>	address of the function calling this kmemory function
<i>bytes_req</i>	requested Bytes
<i>bytes_alloc</i>	allocated Bytes
<i>gfp_flag_name</i>	type of kmemory to allocate(in string format)
<i>ptr</i>	pointer to the kmemory allocated
<i>gfp_flags</i>	type of kmemory to allocate

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>call_site</i>	Address of the function calling this kmemory function
<i>caller_function</i>	Name of the caller function.
<i>name</i>	Name of the probe point

Name

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

Synopsis

`vm.mmap`

Values

address the requested address

name name of the probe point

length the length of the memory segment

Context

The process calling mmap.

Name

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

Synopsis

`vm.munmap`

Values

length the length of the memory segment

address the requested address

name name of the probe point

Context

The process calling munmap.

Name

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

Synopsis

```
vm.oom_kill
```

Values

task the task being killed

name name of the probe point

Context

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

Name

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

Synopsis

vm.pagefault

Values

name name of the probe point

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

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

Context

The process which triggered the fault

Name

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

Synopsis

```
vm.pagefault.return
```

Values

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 name of the probe point

Name

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

Synopsis

```
vm.write_shared
```

Values

address the address of the shared write

name name of the probe point

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

address The address of the shared write

name Name of the probe point

zero boolean indicating whether it is a zero page (can do a clear instead of a copy)

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.write_shared`.

Chapter 7. 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::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::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::cputime_to_usecs — Translates the given cputime into microseconds

Synopsis

```
cputime_to_usecs:long(cputime:long)
```

Arguments

cputime Time to convert to microseconds.

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::nsecs_to_string — Human readable string for given nanoseconds

Synopsis

```
nsecs_to_string:string(nsecs:long)
```

Arguments

nsecs Number of nanoseconds to translate.

Description

Returns a string representing the number of nanoseconds as a human readable string consisting of “XmY.ZZZZZZs”, where X is the number of minutes, Y is the number of seconds and ZZZZZZZZ is the number of nanoseconds.

Name

function::task_start_time — Start time of the given task

Synopsis

```
task_start_time:long(tid:long)
```

Arguments

tid Thread id of the given task

Description

Returns the start time of the given task in nanoseconds since boot time or 0 if the task does not exist.

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_stime_tid — System time of the given task

Synopsis

```
task_stime_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::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”.

Name

function::task_utime — User time of the current task

Synopsis

```
task_utime: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::usecs_to_string — Human readable string for given microseconds

Synopsis

```
usecs_to_string:string(usecs:long)
```

Arguments

usecs Number of microseconds to translate.

Description

Returns a string representing the number of microseconds as a human readable string consisting of “XmY.ZZZZZZs”, where X is the number of minutes, Y is the number of seconds and ZZZZZZ is the number of microseconds.

Chapter 8. Scheduler Tapset

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

Name

probe::scheduler.balance — A cpu attempting to find more work.

Synopsis

```
scheduler.balance
```

Values

name name of the probe point

Context

The cpu looking for more work.

Name

probe::scheduler.cpu_off — Process is about to stop running on a cpu

Synopsis

`scheduler.cpu_off`

Values

<i>name</i>	name of the probe point
<i>idle</i>	boolean indicating whether current is the idle process
<i>task_prev</i>	the process leaving the cpu (same as current)
<i>task_next</i>	the process replacing current

Context

The process leaving the cpu.

Name

probe::scheduler.cpu_on — Process is beginning execution on a cpu

Synopsis

`scheduler.cpu_on`

Values

idle - boolean indicating whether current is the idle process

name name of the probe point

task_prev the process that was previously running on this cpu

Context

The resuming process.

Name

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

Synopsis

`scheduler.ctxswitch`

Values

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

Name

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

Synopsis

```
scheduler.kthread_stop
```

Values

thread_priority priority of the thread

thread_pid 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

name name of the probe point

return_value return value after stopping the thread

Name

probe::scheduler.migrate — Task migrating across cpus

Synopsis

`scheduler.migrate`

Values

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

Name

probe::scheduler.process_exit — Process exiting

Synopsis

```
scheduler.process_exit
```

Values

name name of the probe point

priority priority of the process exiting

pid PID of the process exiting

Name

probe::scheduler.process_fork — Process forked

Synopsis

```
scheduler.process_fork
```

Values

parent_pid PID of the parent process

child_pid PID of the child process

name name of the probe point

Name

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

Synopsis

```
scheduler.process_free
```

Values

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

Name

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

Synopsis

```
scheduler.process_wait
```

Values

name name of the probe point

pid PID of the process scheduler is waiting on

Name

probe::scheduler.signal_send — Sending a signal

Synopsis

```
scheduler.signal_send
```

Values

pid pid of the process sending signal

name name of the probe point

signal_number signal number

Name

probe::scheduler.tick — Scheduler's internal tick, a process's timeslice accounting is updated

Synopsis

```
scheduler.tick
```

Values

name name of the probe point

idle boolean indicating whether current is the idle process

Context

The process whose accounting will be updated.

Name

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

Synopsis

```
scheduler.wait_task
```

Values

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

Name

probe::scheduler.wakeup — Task is woken up

Synopsis

`scheduler.wakeup`

Values

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

Chapter 9. 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::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-of-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.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-of-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_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-of-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.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-of-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.

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

Synopsis

```
ioscheduler.elv_add_request
```

Values

elevator_name The type of I/O elevator currently enabled.

rq_flags Request flags.

disk_minor Disk minor number of request.

rq Address of request.

disk_major Disk major no of request.

q Pointer to request queue.

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>q</i>	pointer to request queue
<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
<i>elevator_name</i>	The type of I/O elevator currently enabled

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

Name

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

Synopsis

```
ioscheduler.elv_next_request
```

Values

elevator_name The type of I/O elevator currently enabled

name Name of the probe point

Name

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

Synopsis

```
ioscheduler.elv_next_request.return
```

Values

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

Name

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

Synopsis

```
ioscheduler_trace.elv_abort_request
```

Values

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

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

Description

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

Name

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

Synopsis

```
ioscheduler_trace.plug
```

Values

name Name of the probe point

rq_queue 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

rq_queue request queue

name Name of the probe point

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

name Name of the probe point

rq_queue request queue

Description

with a request queue expires.

Chapter 10. SCSI Tapset

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

Name

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

Synopsis

`scsi.iocompleted`

Values

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

Name

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

Synopsis

`scsi.iodispatching`

Values

<i>device_state</i>	The current state of the device
<i>lun</i>	The lun number
<i>host_no</i>	The host 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>request_buffer</i>	The request buffer address
<i>channel</i>	The channel number
<i>device_state_str</i>	The current state of the device, as a string
<i>req_addr</i>	The current struct request pointer, as a number
<i>request_bufflen</i>	The request buffer length
<i>data_direction_str</i>	Data direction, as a string
<i>dev_id</i>	The scsi device id

Name

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

Synopsis

`scsi.iodone`

Values

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

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>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)
<i>device_state</i>	The current state of the device

Name

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

Synopsis

```
scsi.ioexecute
```

Values

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

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

Chapter 11. TTY Tapset

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

Name

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

Synopsis

```
tty.init
```

Values

name the driver .dev_name name

driver_name the driver name

module the module name

Name

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

Synopsis

`tty.ioctl`

Values

name the file name

arg the ioctl argument

cmd the ioctl command

Name

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

Synopsis

tty.open

Values

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

Name

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

Synopsis

`tty.poll`

Values

wait_key the wait queue key

file_name the tty file name

Name

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

Synopsis

```
tty.read
```

Values

buffer the buffer that will receive the characters

driver_name the driver name

nr The amount of characters to be read

file_name the file name created to the tty

Name

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

Synopsis

```
tty.receive
```

Values

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

Name

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

Synopsis

```
tty.register
```

Values

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

Name

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

Synopsis

```
tty.release
```

Values

file_flags the file flags

inode_number the inode number

file_name the file name

inode_flags the inode flags

inode_state the inode state

file_mode the file mode

Name

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

Synopsis

```
tty.resize
```

Values

new_ypixel the new ypixel value

old_col the old col value

new_xpixel the new xpixel value

name the tty name

new_col the new col value

new_row the new row value

old_xpixel the old xpixel

old_ypixel the old ypixel

old_row the old row value

Name

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

Synopsis

```
tty.unregister
```

Values

name the driver .dev_name name

driver_name the driver name

module the module name

index the tty index requested

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>nr</i>	The amount of characters
<i>file_name</i>	the file name created to the tty

Chapter 12. 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::irq_handler.entry — Execution of interrupt handler starting

Synopsis

`irq_handler.entry`

Values

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

Name

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

Synopsis

`irq_handler.exit`

Values

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

Name

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

Synopsis

```
softirq.entry
```

Values

vec softirq_action vector

vec_nr softirq vector number

action pointer to softirq handler just about to execute

h struct softirq_action* for current pending softirq

Name

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

Synopsis

```
softirq.exit
```

Values

h struct softirq_action* for just executed softirq

vec softirq_action vector

vec_nr softirq vector number

action pointer to softirq handler that just finished execution

Name

probe::workqueue.create — Creating a new workqueue

Synopsis

`workqueue.create`

Values

wq_thread task_struct of the workqueue thread

cpu cpu for which the worker thread is created

Name

probe::workqueue.destroy — Destroying workqueue

Synopsis

`workqueue.destroy`

Values

wq_thread task_struct of the workqueue thread

Name

probe::workqueue.execute — Executing deferred work

Synopsis

```
workqueue.execute
```

Values

work *work_struct** being executed

wq_thread *task_struct* of the workqueue thread

work_func pointer to handler function

Name

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

Synopsis

```
workqueue.insert
```

Values

wq_thread task_struct of the workqueue thread

work work_struct* being queued

work_func pointer to handler function

Chapter 13. Networking Tapset

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

Name

`function::format_ipaddr` — Returns a string representation for an IP address

Synopsis

```
format_ipaddr:string(addr:long,family:long)
```

Arguments

addr the IP address

family the IP address family (either AF_INET or AF_INET6)

Name

function::htonl — Convert 32-bit long from host to network order

Synopsis

```
htonl:long(x:long)
```

Arguments

x Value to convert

Name

function::htonll — Convert 64-bit long long from host to network order

Synopsis

```
htonll:long(x:long)
```

Arguments

x Value to convert

Name

function::htons — Convert 16-bit short from host to network order

Synopsis

```
htons:long(x:long)
```

Arguments

x Value to convert

Name

function::ip_ntop — Returns a string representation for an IPv4 address

Synopsis

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

Arguments

addr the IPv4 address represented as an integer

Name

function::ntohl — Convert 32-bit long from network to host order

Synopsis

```
ntohl:long(x:long)
```

Arguments

x Value to convert

Name

function::ntohl — Convert 64-bit long long from network to host order

Synopsis

```
ntohl:long(x:long)
```

Arguments

x Value to convert

Name

function::ntohs — Convert 16-bit short from network to host order

Synopsis

```
ntohs:long(x:long)
```

Arguments

x Value to convert

Name

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

Synopsis

```
netdev.change_mac
```

Values

new_mac The new MAC address

mac_len The MAC length

old_mac The current MAC address

dev_name The device that will have the MAC changed

Name

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

Synopsis

```
netdev.change_mtu
```

Values

new_mtu The new MTU

old_mtu The current MTU

dev_name The device that will have the MTU changed

Name

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

Synopsis

```
netdev.change_rx_flag
```

Values

dev_name The device that will be changed

flags The new flags

Name

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

Synopsis

```
netdev.close
```

Values

dev_name The device that is going to be closed

Name

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

Synopsis

```
netdev.get_stats
```

Values

dev_name The device that is going to provide the statistics

Name

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

Synopsis

```
netdev.hard_transmit
```

Values

protocol The protocol used in the transmission

true_size The size of the data to be transmitted.

length The length of the transmit buffer.

dev_name The device scheduled to transmit

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.open — Called when the device is opened

Synopsis

```
netdev.open
```

Values

dev_name The device that is going to be opened

Name

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

Synopsis

```
netdev.receive
```

Values

dev_name The name of the device. e.g: eth0, ath1.

length The length of the receiving buffer.

protocol Protocol of received packet.

Name

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

Synopsis

```
netdev.register
```

Values

dev_name The device that is going to be registered

Name

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

Synopsis

`netdev.rx`

Values

dev_name The device received the packet

protocol The packet protocol

Name

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

Synopsis

```
netdev.set_promiscuity
```

Values

enable If the device is entering promiscuity mode

inc Count the number of promiscuity openers

disable If the device is leaving promiscuity mode

dev_name The device that is entering/leaving promiscuity mode

Name

probe::netdev.transmit — Network device transmitting buffer

Synopsis

```
netdev.transmit
```

Values

protocol The protocol of this packet(defined in include/linux/if_ether.h).

true_size The size of the data to be transmitted.

length The length of the transmit buffer.

dev_name The name of the device. e.g: eth0, ath1.

Name

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

Synopsis

```
netdev.unregister
```

Values

dev_name The device that is going to be unregistered

Name

probe::netfilter.arp.forward — - Called for each ARP packet to be forwarded

Synopsis

```
netfilter.arp.forward
```

Values

<i>arphdr</i>	Address of ARP header
<i>outdev_name</i>	Name of network device packet will be routed to (if known)
<i>nf_stolen</i>	Constant used to signify a 'stolen' verdict
<i>nf_stop</i>	Constant used to signify a 'stop' verdict
<i>nf_queue</i>	Constant used to signify a 'queue' verdict
<i>ar_tip</i>	Ethernet+IP only (ar_pro==0x800): target IP address
<i>nf_repeat</i>	Constant used to signify a 'repeat' verdict
<i>pf</i>	Protocol family -- always “arp”
<i>ar_pro</i>	Format of protocol address
<i>indev_name</i>	Name of network device packet was received on (if known)
<i>ar_hrd</i>	Format of hardware address
<i>nf_drop</i>	Constant used to signify a 'drop' verdict
<i>length</i>	The length of the packet buffer contents, in bytes
<i>indev</i>	Address of net_device representing input device, 0 if unknown
<i>ar_sha</i>	Ethernet+IP only (ar_pro==0x800): source hardware (MAC) address
<i>ar_data</i>	Address of ARP packet data region (after the header)
<i>nf_accept</i>	Constant used to signify an 'accept' verdict
<i>ar_op</i>	ARP opcode (command)
<i>ar_hln</i>	Length of hardware address
<i>outdev</i>	Address of net_device representing output device, 0 if unknown
<i>ar_sip</i>	Ethernet+IP only (ar_pro==0x800): source IP address
<i>ar_tha</i>	Ethernet+IP only (ar_pro==0x800): target hardware (MAC) address
<i>ar_pln</i>	Length of protocol address

Name

probe::netfilter.arp.in — - Called for each incoming ARP packet

Synopsis

netfilter.arp.in

Values

<i>outdev_name</i>	Name of network device packet will be routed to (if known)
<i>arphdr</i>	Address of ARP header
<i>nf_stop</i>	Constant used to signify a 'stop' verdict
<i>nf_stolen</i>	Constant used to signify a 'stolen' verdict
<i>nf_repeat</i>	Constant used to signify a 'repeat' verdict
<i>ar_tip</i>	Ethernet+IP only (ar_pro==0x800): target IP address
<i>nf_queue</i>	Constant used to signify a 'queue' verdict
<i>indev_name</i>	Name of network device packet was received on (if known)
<i>ar_pro</i>	Format of protocol address
<i>pf</i>	Protocol family -- always "arp"
<i>nf_drop</i>	Constant used to signify a 'drop' verdict
<i>ar_hrd</i>	Format of hardware address
<i>nf_accept</i>	Constant used to signify an 'accept' verdict
<i>ar_data</i>	Address of ARP packet data region (after the header)
<i>ar_sha</i>	Ethernet+IP only (ar_pro==0x800): source hardware (MAC) address
<i>length</i>	The length of the packet buffer contents, in bytes
<i>indev</i>	Address of net_device representing input device, 0 if unknown
<i>ar_hln</i>	Length of hardware address
<i>ar_op</i>	ARP opcode (command)
<i>ar_tha</i>	Ethernet+IP only (ar_pro==0x800): target hardware (MAC) address
<i>ar_pln</i>	Length of protocol address
<i>ar_sip</i>	Ethernet+IP only (ar_pro==0x800): source IP address
<i>outdev</i>	Address of net_device representing output device, 0 if unknown

Name

probe::netfilter.arp.out — - Called for each outgoing ARP packet

Synopsis

netfilter.arp.out

Values

<i>ar_tip</i>	Ethernet+IP only (ar_pro==0x800): target IP address
<i>nf_queue</i>	Constant used to signify a 'queue' verdict
<i>nf_repeat</i>	Constant used to signify a 'repeat' verdict
<i>ar_pro</i>	Format of protocol address
<i>pf</i>	Protocol family -- always “arp”
<i>indev_name</i>	Name of network device packet was received on (if known)
<i>outdev_name</i>	Name of network device packet will be routed to (if known)
<i>arphdr</i>	Address of ARP header
<i>nf_stolen</i>	Constant used to signify a 'stolen' verdict
<i>nf_stop</i>	Constant used to signify a 'stop' verdict
<i>ar_op</i>	ARP opcode (command)
<i>ar_hln</i>	Length of hardware address
<i>outdev</i>	Address of net_device representing output device, 0 if unknown
<i>ar_tha</i>	Ethernet+IP only (ar_pro==0x800): target hardware (MAC) address
<i>ar_pln</i>	Length of protocol address
<i>ar_sip</i>	Ethernet+IP only (ar_pro==0x800): source IP address
<i>ar_hrd</i>	Format of hardware address
<i>nf_drop</i>	Constant used to signify a 'drop' verdict
<i>ar_sha</i>	Ethernet+IP only (ar_pro==0x800): source hardware (MAC) address
<i>length</i>	The length of the packet buffer contents, in bytes
<i>indev</i>	Address of net_device representing input device, 0 if unknown
<i>nf_accept</i>	Constant used to signify an 'accept' verdict
<i>ar_data</i>	Address of ARP packet data region (after the header)

Name

probe::netfilter.bridge.forward — Called on an incoming bridging packet destined for some other computer

Synopsis

```
netfilter.bridge.forward
```

Values

<i>br_rid</i>	Identity of root bridge
<i>nf_drop</i>	Constant used to signify a 'drop' verdict
<i>br_hitime</i>	Hello time in 1/256 secs
<i>nf_accept</i>	Constant used to signify an 'accept' verdict
<i>br_fd</i>	Forward delay in 1/256 secs
<i>length</i>	The length of the packet buffer contents, in bytes
<i>indev</i>	Address of net_device representing input device, 0 if unknown
<i>br_prid</i>	Protocol identifier
<i>br_type</i>	BPDU type
<i>outdev</i>	Address of net_device representing output device, 0 if unknown
<i>br_max</i>	Max age in 1/256 secs
<i>llcpdu</i>	Address of LLC Protocol Data Unit
<i>protocol</i>	Packet protocol
<i>br_flags</i>	BPDU flags
<i>br_bid</i>	Identity of bridge
<i>br_cost</i>	Total cost from transmitting bridge to root
<i>outdev_name</i>	Name of network device packet will be routed to (if known)
<i>brhdr</i>	Address of bridge header
<i>br_msg</i>	Message age in 1/256 secs
<i>nf_stop</i>	Constant used to signify a 'stop' verdict
<i>br_poid</i>	Port identifier
<i>nf_stolen</i>	Constant used to signify a 'stolen' verdict
<i>nf_repeat</i>	Constant used to signify a 'repeat' verdict
<i>nf_queue</i>	Constant used to signify a 'queue' verdict
<i>br_rmac</i>	Root bridge MAC address
<i>llcproto_stp</i>	Constant used to signify Bridge Spanning Tree Protocol packet

<i>br_vid</i>	Protocol version identifier
<i>indev_name</i>	Name of network device packet was received on (if known)
<i>pf</i>	Protocol family -- always “bridge”
<i>br_mac</i>	Bridge MAC address

Name

probe::netfilter.bridge.local_in — Called on a bridging packet destined for the local computer

Synopsis

netfilter.bridge.local_in

Values

<i>nf_repeat</i>	Constant used to signify a 'repeat' verdict
<i>nf_queue</i>	Constant used to signify a 'queue' verdict
<i>br_rmac</i>	Root bridge MAC address
<i>llcproto_stp</i>	Constant used to signify Bridge Spanning Tree Protocol packet
<i>indev_name</i>	Name of network device packet was received on (if known)
<i>br_vid</i>	Protocol version identifier
<i>pf</i>	Protocol family -- always “bridge”
<i>br_mac</i>	Bridge MAC address
<i>br_bid</i>	Identity of bridge
<i>br_flags</i>	BPDU flags
<i>br_cost</i>	Total cost from transmitting bridge to root
<i>outdev_name</i>	Name of network device packet will be routed to (if known)
<i>brhdr</i>	Address of bridge header
<i>br_msg</i>	Message age in 1/256 secs
<i>nf_stop</i>	Constant used to signify a 'stop' verdict
<i>br_poid</i>	Port identifier
<i>nf_stolen</i>	Constant used to signify a 'stolen' verdict
<i>br_type</i>	BPDU type
<i>outdev</i>	Address of net_device representing output device, 0 if unknown
<i>protocol</i>	Packet protocol
<i>br_max</i>	Max age in 1/256 secs
<i>llcpdu</i>	Address of LLC Protocol Data Unit
<i>br_rid</i>	Identity of root bridge
<i>nf_drop</i>	Constant used to signify a 'drop' verdict
<i>br_hitime</i>	Hello time in 1/256 secs
<i>br_fd</i>	Forward delay in 1/256 secs

<i>nf_accept</i>	Constant used to signify an 'accept' verdict
<i>indev</i>	Address of net_device representing input device, 0 if unknown
<i>length</i>	The length of the packet buffer contents, in bytes
<i>br_prid</i>	Protocol identifier

Name

probe::netfilter.bridge.local_out — Called on a bridging packet coming from a local process

Synopsis

netfilter.bridge.local_out

Values

<i>br_mac</i>	Bridge MAC address
<i>pf</i>	Protocol family -- always “bridge”
<i>br_vid</i>	Protocol version identifier
<i>indev_name</i>	Name of network device packet was received on (if known)
<i>llcproto_stp</i>	Constant used to signify Bridge Spanning Tree Protocol packet
<i>nf_queue</i>	Constant used to signify a 'queue' verdict
<i>br_rmac</i>	Root bridge MAC address
<i>nf_repeat</i>	Constant used to signify a 'repeat' verdict
<i>nf_stolen</i>	Constant used to signify a 'stolen' verdict
<i>br_poid</i>	Port identifier
<i>nf_stop</i>	Constant used to signify a 'stop' verdict
<i>br_msg</i>	Message age in 1/256 secs
<i>brhdr</i>	Address of bridge header
<i>outdev_name</i>	Name of network device packet will be routed to (if known)
<i>br_bid</i>	Identity of bridge
<i>br_flags</i>	BPDU flags
<i>br_cost</i>	Total cost from transmitting bridge to root
<i>br_max</i>	Max age in 1/256 secs
<i>protocol</i>	Packet protocol
<i>llcpdu</i>	Address of LLC Protocol Data Unit
<i>outdev</i>	Address of net_device representing output device, 0 if unknown
<i>br_type</i>	BPDU type
<i>br_prid</i>	Protocol identifier
<i>length</i>	The length of the packet buffer contents, in bytes
<i>indev</i>	Address of net_device representing input device, 0 if unknown
<i>nf_accept</i>	Constant used to signify an 'accept' verdict

<i>br_fd</i>	Forward delay in 1/256 secs
<i>br_hptime</i>	Hello time in 1/256 secs
<i>nf_drop</i>	Constant used to signify a 'drop' verdict
<i>br_rid</i>	Identity of root bridge

Name

probe::netfilter.bridge.post_routing — - Called before a bridging packet hits the wire

Synopsis

netfilter.bridge.post_routing

Values

<i>br_fd</i>	Forward delay in 1/256 secs
<i>nf_accept</i>	Constant used to signify an 'accept' verdict
<i>length</i>	The length of the packet buffer contents, in bytes
<i>indev</i>	Address of net_device representing input device, 0 if unknown
<i>br_prid</i>	Protocol identifier
<i>nf_drop</i>	Constant used to signify a 'drop' verdict
<i>br_rid</i>	Identity of root bridge
<i>br_htime</i>	Hello time in 1/256 secs
<i>br_max</i>	Max age in 1/256 secs
<i>protocol</i>	Packet protocol
<i>llcpdu</i>	Address of LLC Protocol Data Unit
<i>outdev</i>	Address of net_device representing output device, 0 if unknown
<i>br_type</i>	BPDU type
<i>nf_stop</i>	Constant used to signify a 'stop' verdict
<i>br_poid</i>	Port identifier
<i>nf_stolen</i>	Constant used to signify a 'stolen' verdict
<i>brhdr</i>	Address of bridge header
<i>outdev_name</i>	Name of network device packet will be routed to (if known)
<i>br_bid</i>	Identity of bridge
<i>br_cost</i>	Total cost from transmitting bridge to root
<i>br_flags</i>	BPDU flags
<i>br_msg</i>	Message age in 1/256 secs
<i>br_vid</i>	Protocol version identifier
<i>indev_name</i>	Name of network device packet was received on (if known)
<i>llcproto_stp</i>	Constant used to signify Bridge Spanning Tree Protocol packet
<i>br_mac</i>	Bridge MAC address

<i>pf</i>	Protocol family -- always "bridge"
<i>nf_repeat</i>	Constant used to signify a 'repeat' verdict
<i>nf_queue</i>	Constant used to signify a 'queue' verdict
<i>br_rmac</i>	Root bridge MAC address

Name

probe::netfilter.bridge.pre_routing — - Called before a bridging packet is routed

Synopsis

netfilter.bridge.pre_routing

Values

<i>br_msg</i>	Message age in 1/256 secs
<i>br_bid</i>	Identity of bridge
<i>br_flags</i>	BPDU flags
<i>br_cost</i>	Total cost from transmitting bridge to root
<i>outdev_name</i>	Name of network device packet will be routed to (if known)
<i>brhdr</i>	Address of bridge header
<i>nf_stolen</i>	Constant used to signify a 'stolen' verdict
<i>br_poid</i>	Port identifier
<i>nf_stop</i>	Constant used to signify a 'stop' verdict
<i>nf_queue</i>	Constant used to signify a 'queue' verdict
<i>br_rmac</i>	Root bridge MAC address
<i>nf_repeat</i>	Constant used to signify a 'repeat' verdict
<i>pf</i>	Protocol family -- always “bridge”
<i>br_mac</i>	Bridge MAC address
<i>llcproto_stp</i>	Constant used to signify Bridge Spanning Tree Protocol packet
<i>br_vid</i>	Protocol version identifier
<i>indev_name</i>	Name of network device packet was received on (if known)
<i>br_hitime</i>	Hello time in 1/256 secs
<i>br_rid</i>	Identity of root bridge
<i>nf_drop</i>	Constant used to signify a 'drop' verdict
<i>br_prid</i>	Protocol identifier
<i>length</i>	The length of the packet buffer contents, in bytes
<i>indev</i>	Address of net_device representing input device, 0 if unknown
<i>nf_accept</i>	Constant used to signify an 'accept' verdict
<i>br_fd</i>	Forward delay in 1/256 secs
<i>br_type</i>	BPDU type

<i>outdev</i>	Address of net_device representing output device, 0 if unknown
<i>llcpdu</i>	Address of LLC Protocol Data Unit
<i>br_max</i>	Max age in 1/256 secs
<i>protocol</i>	Packet protocol

Name

probe::netfilter.ip.forward — Called on an incoming IP packet addressed to some other computer

Synopsis

`netfilter.ip.forward`

Values

<i>pf</i>	Protocol family -- either “ipv4” or “ipv6”
<i>iphdr</i>	Address of IP header
<i>indev_name</i>	Name of network device packet was received on (if known)
<i>psb</i>	TCP PSB flag (if protocol is TCP; ipv4 only)
<i>nf_queue</i>	Constant used to signify a 'queue' verdict
<i>rst</i>	TCP RST flag (if protocol is TCP; ipv4 only)
<i>syn</i>	TCP SYN flag (if protocol is TCP; ipv4 only)
<i>nf_repeat</i>	Constant used to signify a 'repeat' verdict
<i>nf_stolen</i>	Constant used to signify a 'stolen' verdict
<i>ipproto_tcp</i>	Constant used to signify that the packet protocol is TCP
<i>fin</i>	TCP FIN flag (if protocol is TCP; ipv4 only)
<i>nf_stop</i>	Constant used to signify a 'stop' verdict
<i>saddr</i>	A string representing the source IP address
<i>outdev_name</i>	Name of network device packet will be routed to (if known)
<i>daddr</i>	A string representing the destination IP address
<i>protocol</i>	Packet protocol from driver (ipv4 only)
<i>family</i>	IP address family
<i>outdev</i>	Address of net_device representing output device, 0 if unknown
<i>urg</i>	TCP URG flag (if protocol is TCP; ipv4 only)
<i>ack</i>	TCP ACK flag (if protocol is TCP; ipv4 only)
<i>ipproto_udp</i>	Constant used to signify that the packet protocol is UDP
<i>length</i>	The length of the packet buffer contents, in bytes
<i>indev</i>	Address of net_device representing input device, 0 if unknown
<i>dport</i>	TCP or UDP destination port (ipv4 only)
<i>nf_accept</i>	Constant used to signify an 'accept' verdict
<i>sport</i>	TCP or UDP source port (ipv4 only)

nf_drop Constant used to signify a 'drop' verdict

Name

probe::netfilter.ip.local_in — Called on an incoming IP packet addressed to the local computer

Synopsis

netfilter.ip.local_in

Values

<i>nf_stop</i>	Constant used to signify a 'stop' verdict
<i>fin</i>	TCP FIN flag (if protocol is TCP; ipv4 only)
<i>ipproto_tcp</i>	Constant used to signify that the packet protocol is TCP
<i>nf_stolen</i>	Constant used to signify a 'stolen' verdict
<i>daddr</i>	A string representing the destination IP address
<i>outdev_name</i>	Name of network device packet will be routed to (if known)
<i>saddr</i>	A string representing the source IP address
<i>indev_name</i>	Name of network device packet was received on (if known)
<i>iphdr</i>	Address of IP header
<i>pf</i>	Protocol family -- either “ipv4” or “ipv6”
<i>nf_repeat</i>	Constant used to signify a 'repeat' verdict
<i>rst</i>	TCP RST flag (if protocol is TCP; ipv4 only)
<i>syn</i>	TCP SYN flag (if protocol is TCP; ipv4 only)
<i>nf_queue</i>	Constant used to signify a 'queue' verdict
<i>psb</i>	TCP PSB flag (if protocol is TCP; ipv4 only)
<i>sport</i>	TCP or UDP source port (ipv4 only)
<i>nf_accept</i>	Constant used to signify an 'accept' verdict
<i>dport</i>	TCP or UDP destination port (ipv4 only)
<i>indev</i>	Address of net_device representing input device, 0 if unknown
<i>length</i>	The length of the packet buffer contents, in bytes
<i>nf_drop</i>	Constant used to signify a 'drop' verdict
<i>family</i>	IP address family
<i>outdev</i>	Address of net_device representing output device, 0 if unknown
<i>protocol</i>	Packet protocol from driver (ipv4 only)
<i>ack</i>	TCP ACK flag (if protocol is TCP; ipv4 only)
<i>ipproto_udp</i>	Constant used to signify that the packet protocol is UDP

urg TCP URG flag (if protocol is TCP; ipv4 only)

Name

probe::netfilter.ip.local_out — Called on an outgoing IP packet

Synopsis

netfilter.ip.local_out

Values

<i>psh</i>	TCP PSH flag (if protocol is TCP; ipv4 only)
<i>nf_queue</i>	Constant used to signify a 'queue' verdict
<i>rst</i>	TCP RST flag (if protocol is TCP; ipv4 only)
<i>syn</i>	TCP SYN flag (if protocol is TCP; ipv4 only)
<i>nf_repeat</i>	Constant used to signify a 'repeat' verdict
<i>iphdr</i>	Address of IP header
<i>pf</i>	Protocol family -- either “ipv4” or “ipv6”
<i>indev_name</i>	Name of network device packet was received on (if known)
<i>saddr</i>	A string representing the source IP address
<i>outdev_name</i>	Name of network device packet will be routed to (if known)
<i>daddr</i>	A string representing the destination IP address
<i>nf_stolen</i>	Constant used to signify a 'stolen' verdict
<i>ipproto_tcp</i>	Constant used to signify that the packet protocol is TCP
<i>fin</i>	TCP FIN flag (if protocol is TCP; ipv4 only)
<i>nf_stop</i>	Constant used to signify a 'stop' verdict
<i>urg</i>	TCP URG flag (if protocol is TCP; ipv4 only)
<i>ack</i>	TCP ACK flag (if protocol is TCP; ipv4 only)
<i>ipproto_udp</i>	Constant used to signify that the packet protocol is UDP
<i>protocol</i>	Packet protocol from driver (ipv4 only)
<i>family</i>	IP address family
<i>outdev</i>	Address of net_device representing output device, 0 if unknown
<i>nf_drop</i>	Constant used to signify a 'drop' verdict
<i>length</i>	The length of the packet buffer contents, in bytes
<i>indev</i>	Address of net_device representing input device, 0 if unknown
<i>dport</i>	TCP or UDP destination port (ipv4 only)
<i>nf_accept</i>	Constant used to signify an 'accept' verdict

sport TCP or UDP source port (ipv4 only)

Name

probe::netfilter.ip.post_routing — Called immediately before an outgoing IP packet leaves the computer

Synopsis

`netfilter.ip.post_routing`

Values

<i>outdev</i>	Address of net_device representing output device, 0 if unknown
<i>family</i>	IP address family
<i>protocol</i>	Packet protocol from driver (ipv4 only)
<i>ack</i>	TCP ACK flag (if protocol is TCP; ipv4 only)
<i>ipproto_udp</i>	Constant used to signify that the packet protocol is UDP
<i>urg</i>	TCP URG flag (if protocol is TCP; ipv4 only)
<i>sport</i>	TCP or UDP source port (ipv4 only)
<i>dport</i>	TCP or UDP destination port (ipv4 only)
<i>nf_accept</i>	Constant used to signify an 'accept' verdict
<i>indev</i>	Address of net_device representing input device, 0 if unknown
<i>length</i>	The length of the packet buffer contents, in bytes
<i>nf_drop</i>	Constant used to signify a 'drop' verdict
<i>indev_name</i>	Name of network device packet was received on (if known)
<i>pf</i>	Protocol family -- either “ipv4” or “ipv6”
<i>iphdr</i>	Address of IP header
<i>nf_repeat</i>	Constant used to signify a 'repeat' verdict
<i>syn</i>	TCP SYN flag (if protocol is TCP; ipv4 only)
<i>rst</i>	TCP RST flag (if protocol is TCP; ipv4 only)
<i>nf_queue</i>	Constant used to signify a 'queue' verdict
<i>psh</i>	TCP PSH flag (if protocol is TCP; ipv4 only)
<i>nf_stop</i>	Constant used to signify a 'stop' verdict
<i>fin</i>	TCP FIN flag (if protocol is TCP; ipv4 only)
<i>ipproto_tcp</i>	Constant used to signify that the packet protocol is TCP
<i>nf_stolen</i>	Constant used to signify a 'stolen' verdict
<i>daddr</i>	A string representing the destination IP address
<i>outdev_name</i>	Name of network device packet will be routed to (if known)

saddr A string representing the source IP address

Name

probe::netfilter.ip.pre_routing — Called before an IP packet is routed

Synopsis

netfilter.ip.pre_routing

Values

<i>outdev</i>	Address of net_device representing output device, 0 if unknown
<i>family</i>	IP address family
<i>protocol</i>	Packet protocol from driver (ipv4 only)
<i>urg</i>	TCP URG flag (if protocol is TCP; ipv4 only)
<i>ack</i>	TCP ACK flag (if protocol is TCP; ipv4 only)
<i>ipproto_udp</i>	Constant used to signify that the packet protocol is UDP
<i>length</i>	The length of the packet buffer contents, in bytes
<i>indev</i>	Address of net_device representing input device, 0 if unknown
<i>sport</i>	TCP or UDP source port (ipv4 only)
<i>dport</i>	TCP or UDP destination port (ipv4 only)
<i>nf_accept</i>	Constant used to signify an 'accept' verdict
<i>nf_drop</i>	Constant used to signify a 'drop' verdict
<i>pf</i>	Protocol family - either 'ipv4' or 'ipv6'
<i>iphdr</i>	Address of IP header
<i>indev_name</i>	Name of network device packet was received on (if known)
<i>nf_queue</i>	Constant used to signify a 'queue' verdict
<i>psh</i>	TCP PSH flag (if protocol is TCP; ipv4 only)
<i>nf_repeat</i>	Constant used to signify a 'repeat' verdict
<i>rst</i>	TCP RST flag (if protocol is TCP; ipv4 only)
<i>syn</i>	TCP SYN flag (if protocol is TCP; ipv4 only)
<i>nf_stolen</i>	Constant used to signify a 'stolen' verdict
<i>nf_stop</i>	Constant used to signify a 'stop' verdict
<i>fin</i>	TCP FIN flag (if protocol is TCP; ipv4 only)
<i>ipproto_tcp</i>	Constant used to signify that the packet protocol is TCP
<i>saddr</i>	A string representing the source IP address
<i>daddr</i>	A string representing the destination IP address

outdev_name Name of network device packet will be routed to (if known)

Name

probe::sunrpc.clnt.bind_new_program — Bind a new RPC program to an existing client

Synopsis

```
sunrpc.clnt.bind_new_program
```

Values

<i>vers</i>	the version of new RPC program
<i>progrname</i>	the name of new RPC program
<i>old_prog</i>	the number of old RPC program
<i>old_vers</i>	the version of old RPC program
<i>prog</i>	the number of new RPC program
<i>old_progrname</i>	the name of old RPC program
<i>servername</i>	the server machine name

Name

probe::sunrpc.clnt.call_async — Make an asynchronous RPC call

Synopsis

```
sunrpc.clnt.call_async
```

Values

<i>flags</i>	flags
<i>xid</i>	current transmission id
<i>dead</i>	whether this client is abandoned
<i>progname</i>	the RPC program name
<i>vers</i>	the RPC program version number
<i>procname</i>	the procedure name in this RPC call
<i>servername</i>	the server machine name
<i>prog</i>	the RPC program number
<i>proc</i>	the procedure number in this RPC call
<i>port</i>	the port number
<i>prot</i>	the IP protocol number

Name

probe::sunrpc.clnt.call_sync — Make a synchronous RPC call

Synopsis

```
sunrpc.clnt.call_sync
```

Values

<i>port</i>	the port number
<i>prot</i>	the IP protocol number
<i>servername</i>	the server machine name
<i>procname</i>	the procedure name in this RPC call
<i>proc</i>	the procedure number in this RPC call
<i>prog</i>	the RPC program number
<i>dead</i>	whether this client is abandoned
<i>progrname</i>	the RPC program name
<i>vers</i>	the RPC program version number
<i>flags</i>	flags
<i>xid</i>	current transmission id

Name

probe::sunrpc.clnt.clone_client — Clone an RPC client structure

Synopsis

```
sunrpc.clnt.clone_client
```

Values

<i>servername</i>	the server machine name
<i>port</i>	the port number
<i>progname</i>	the RPC program name
<i>vers</i>	the RPC program version number
<i>prog</i>	the RPC program number
<i>prot</i>	the IP protocol number
<i>authflavor</i>	the authentication flavor

Name

probe::sunrpc.clnt.create_client — Create an RPC client

Synopsis

```
sunrpc.clnt.create_client
```

Values

<i>port</i>	the port number
<i>servername</i>	the server machine name
<i>vers</i>	the RPC program version number
<i>progname</i>	the RPC program name
<i>authflavor</i>	the authentication flavor
<i>prot</i>	the IP protocol number
<i>prog</i>	the RPC program number

Name

probe::sunrpc.clnt.restart_call — Restart an asynchronous RPC call

Synopsis

```
sunrpc.clnt.restart_call
```

Values

<i>xid</i>	the transmission id
<i>tk_priority</i>	the task priority
<i>servername</i>	the server machine name
<i>prog</i>	the RPC program number
<i>tk_runstate</i>	the task run status
<i>tk_flags</i>	the task flags
<i>tk_pid</i>	the debugging aid of task

Name

probe::sunrpc.clnt.shutdown_client — Shutdown an RPC client

Synopsis

`sunrpc.clnt.shutdown_client`

Values

<i>prot</i>	the IP protocol number
<i>om_execute</i>	the RPC execution jiffies
<i>om_queue</i>	the jiffies queued for xmit
<i>port</i>	the port number
<i>netreconn</i>	the count of reconnections
<i>om_ntrans</i>	the count of RPC transmissions
<i>clones</i>	the number of clones
<i>prog</i>	the RPC program number
<i>rpccnt</i>	the count of RPC calls
<i>servername</i>	the server machine name
<i>authflavor</i>	the authentication flavor
<i>om_bytes_recv</i>	the count of bytes in
<i>om_bytes_sent</i>	the count of bytes out
<i>tasks</i>	the number of references
<i>om_rtt</i>	the RPC RTT jiffies
<i>vers</i>	the RPC program version number
<i>progrname</i>	the RPC program name
<i>om_ops</i>	the count of operations

Name

probe::sunrpc.sched.delay — Delay an RPC task

Synopsis

`sunrpc.sched.delay`

Values

<i>delay</i>	the time delayed
<i>xid</i>	the transmission id in the RPC call
<i>vers</i>	the program version in the RPC call
<i>prog</i>	the program number in the RPC call
<i>tk_flags</i>	the flags of the task
<i>tk_pid</i>	the debugging id of the task
<i>prot</i>	the IP protocol in the RPC call

Name

probe::sunrpc.sched.execute — Execute the RPC `scheduler'

Synopsis

`sunrpc.sched.execute`

Values

<i>prog</i>	the program number in the RPC call
<i>tk_pid</i>	the debugging id of the task
<i>prot</i>	the IP protocol in the RPC call
<i>tk_flags</i>	the flags of the task
<i>xid</i>	the transmission id in the RPC call
<i>vers</i>	the program version in the RPC call

Name

probe::sunrpc.sched.new_task — Create new task for the specified client

Synopsis

`sunrpc.sched.new_task`

Values

vers the program version in the RPC call

xid the transmission id in the RPC call

prot the IP protocol in the RPC call

tk_flags the flags of the task

prog the program number in the RPC call

Name

probe::sunrpc.sched.release_task — Release all resources associated with a task

Synopsis

```
sunrpc.sched.release_task
```

Values

tk_flags the flags of the task

prot the IP protocol in the RPC call

prog the program number in the RPC call

vers the program version in the RPC call

xid the transmission id in the RPC call

Description

`rpc_release_task` function might not be found for a particular kernel. So, if we can't find it, just return '-1' for everything.

Name

probe::sunrpc.svc.create — Create an RPC service

Synopsis

```
sunrpc.svc.create
```

Values

progname the name of the program

bufsize the buffer size

prog the number of the program

pg_nvers the number of supported versions

Name

probe::sunrpc.svc.destroy — Destroy an RPC service

Synopsis

```
sunrpc.svc.destroy
```

Values

<i>sv_progname</i>	the name of the program
<i>nettcpconn</i>	the count of accepted TCP connections
<i>sv_prog</i>	the number of the program
<i>sv_name</i>	the service name
<i>sv_nrthreads</i>	the number of concurrent threads
<i>rpcbadfmt</i>	the count of requests dropped for bad formats
<i>netcnt</i>	the count of received RPC requests
<i>rpccnt</i>	the count of valid RPC requests
<i>rpcbadauth</i>	the count of requests drooped for authentication failure

Name

probe::sunrpc.svc.drop — Drop RPC request

Synopsis

`sunrpc.svc.drop`

Values

rq_vers the program version in the request

rq_prog the program number in the request

sv_name the service name

rq_prot the IP protocol of the request

rq_proc the procedure number in the request

peer_ip the peer address where the request is from

rq_xid the transmission id in the request

Name

probe::sunrpc.svc.process — Process an RPC request

Synopsis

`sunrpc.svc.process`

Values

<i>peer_ip</i>	the peer address where the request is from
<i>rq_proc</i>	the procedure number in the request
<i>rq_xid</i>	the transmission id in the request
<i>sv_nrthreads</i>	the number of concurrent threads
<i>sv_prog</i>	the number of the program
<i>sv_name</i>	the service name
<i>rq_vers</i>	the program version in the request
<i>rq_prog</i>	the program number in the request
<i>rq_prot</i>	the IP protocol of the request

Name

probe::sunrpc.svc.recv — Listen for the next RPC request on any socket

Synopsis

```
sunrpc.svc.recv
```

Values

<i>sv_name</i>	the service name
<i>sv_prog</i>	the number of the program
<i>sv_nrthreads</i>	the number of concurrent threads
<i>timeout</i>	the timeout of waiting for data

Name

probe::sunrpc.svc.register — Register an RPC service with the local portmapper

Synopsis

```
sunrpc.svc.register
```

Values

sv_name the service name

progname the name of the program

prog the number of the program

port the port number

prot the IP protocol number

Description

If *proto* and *port* are both 0, then unregister a service.

Name

probe::sunrpc.svc.send — Return reply to RPC client

Synopsis

```
sunrpc.svc.send
```

Values

sv_name the service name

rq_vers the program version in the request

rq_prog the program number in the request

rq_prot the IP protocol of the request

peer_ip the peer address where the request is from

rq_proc the procedure number in the request

rq_xid the transmission id in the request

Name

probe::tcp.disconnect — TCP socket disconnection

Synopsis

`tcp.disconnect`

Values

flags TCP flags (e.g. FIN, etc)

daddr A string representing the destination IP address

name Name of this probe

sport TCP source port

sock Network socket

dport TCP destination port

saddr A string representing the source IP address

family IP address family

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.receive — Called when a TCP packet is received

Synopsis

tcp.receive

Values

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

Name

probe::tcp.recvmsg — Receiving TCP message

Synopsis

`tcp.recvmsg`

Values

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

Context

The process which receives a tcp message

Name

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

Synopsis

```
tcp.recvmsg.return
```

Values

dport TCP destination port

family IP address family

saddr A string representing the source IP address

sport TCP source port

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

daddr A string representing the destination IP address

name Name of this probe

Context

The process which receives a tcp message

Name

probe::tcp.sendmsg — Sending a tcp message

Synopsis

`tcp.sendmsg`

Values

sock Network socket

size Number of bytes to send

name Name of this probe

family IP address family

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.setsockopt — Call to setsockopt

Synopsis

`tcp.setsockopt`

Values

<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
<i>optstr</i>	Resolves optname to a human-readable format
<i>family</i>	IP address family

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

Synopsis

udp.disconnect

Values

<i>sock</i>	Network socket used by the process
<i>dport</i>	UDP destination port
<i>saddr</i>	A string representing the source IP address
<i>sport</i>	UDP source port
<i>daddr</i>	A string representing the destination IP address
<i>flags</i>	Flags (e.g. FIN, etc)
<i>name</i>	The name of this probe
<i>family</i>	IP address family

Context

The process which requests a UDP disconnection

Name

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

Synopsis

```
udp.disconnect.return
```

Values

dport UDP destination port

saddr A string representing the source IP address

sport UDP source port

daddr A string representing the destination IP address

name The name of this probe

family IP address family

ret Error code (0: no error)

Context

The process which requested a UDP disconnection

Name

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

Synopsis

`udp.recvmsg`

Values

daddr A string representing the destination IP address

saddr A string representing the source IP address

sport UDP source port

dport UDP destination port

sock Network socket used by the process

family IP address family

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

Synopsis

```
udp.recvmsg.return
```

Values

<i>family</i>	IP address family
<i>size</i>	Number of bytes received by the process
<i>name</i>	The name of this probe
<i>dport</i>	UDP destination port
<i>daddr</i>	A string representing the destination IP address
<i>sport</i>	UDP source port
<i>saddr</i>	A string representing the source IP address

Context

The process which received a UDP message

Name

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

Synopsis

`udp.sendmsg`

Values

<i>sock</i>	Network socket used by the process
<i>dport</i>	UDP destination port
<i>sport</i>	UDP source port
<i>saddr</i>	A string representing the source IP address
<i>daddr</i>	A string representing the destination IP address
<i>size</i>	Number of bytes sent by the process
<i>name</i>	The name of this probe
<i>family</i>	IP address family

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

name The name of this probe

size Number of bytes sent by the process

Context

The process which sent a UDP message

Chapter 14. Socket Tapset

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

Name

function::inet_get_ip_source — Provide IP source address string for a kernel socket

Synopsis

```
inet_get_ip_source:string(sock:long)
```

Arguments

sock pointer to the kernel socket

Name

function::inet_get_local_port — Provide local port number for a kernel socket

Synopsis

```
inet_get_local_port:long(sock:long)
```

Arguments

sock pointer to the kernel socket

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 protocol family number

Synopsis

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

Arguments

family The family name

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_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

Name

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

Synopsis

`socket.aio_read`

Values

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

Synopsis

```
socket.aio_write
```

Values

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

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.close — Close a socket

Synopsis

```
socket.close
```

Values

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

probe::socket.create — Creation of a socket

Synopsis

```
socket.create
```

Values

protocol Protocol value

type Socket type value

requester Requested by user process or the kernel (1 = kernel, 0 = user)

family Protocol family value

name Name of this probe

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

Synopsis

```
socket.read_iter
```

Values

family Protocol family value

state Socket state value

name Name of this probe

flags Socket flags value

size Message size in bytes

type Socket type value

protocol Protocol value

Context

The message sender

Description

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

Name

probe::socket.read_iter.return — Conclusion of message received via `sock_read_iter`

Synopsis

`socket.read_iter.return`

Values

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

Context

The message receiver.

Description

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

Name

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

Synopsis

`socket.readv`

Values

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

Context

The message receiver.

Description

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

Name

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

Synopsis

```
socket.receive
```

Values

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

Context

The message receiver

Name

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

Synopsis

`socket.recvmsg`

Values

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

Synopsis

`socket.send`

Values

protocol Protocol value

type Socket type value

success Was send successful? (1 = yes, 0 = no)

size Size of message sent (in bytes) or error code if success = 0

flags Socket flags value

name Name of this probe

state Socket state value

family Protocol family value

Context

The message sender

Name

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

Synopsis

`socket.sendmsg`

Values

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

Synopsis

```
socket.write_iter
```

Values

<i>size</i>	Message size in bytes
<i>type</i>	Socket type value
<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>family</i>	Protocol family value

Context

The message sender

Description

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

Name

probe::socket.write_iter.return — Conclusion of message send via sock_write_iter

Synopsis

```
socket.write_iter.return
```

Values

<i>name</i>	Name of this probe
<i>family</i>	Protocol family value
<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>protocol</i>	Protocol value
<i>success</i>	Was receive successful? (1 = yes, 0 = no)
<i>flags</i>	Socket flags value

Context

The message receiver.

Description

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

Name

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

Synopsis

`socket.writev`

Values

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

Context

The message receiver.

Description

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

Chapter 15. 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_filter_key — Default filter function for ipmib.* probes

Synopsis

```
ipmib_filter_key:long(skb:long,op:long,SourceIsLocal:long)
```

Arguments

skb pointer to the struct sk_buff

op value to be counted if *skb* passes the filter

SourceIsLocal 1 is local operation and 0 is non-local operation

Description

This function is a default filter function. The user can replace this function with their own. The user-supplied filter function returns an index key based on the values in *skb*. A return value of 0 means this particular *skb* should be not be counted.

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

function::ipmib_local_addr — Get the local ip address

Synopsis

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

Arguments

skb pointer to a struct sk_buff

SourceIsLocal flag to indicate whether local operation

Description

Returns the local ip address *skb*.

Name

function::ipmib_remote_addr — Get the remote ip address

Synopsis

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

Arguments

skb pointer to a struct *sk_buff*

SourceIsLocal flag to indicate whether local operation

Description

Returns the remote ip address from *skb*.

Name

function::ipmib_tcp_local_port — Get the local tcp port

Synopsis

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

Arguments

skb pointer to a struct sk_buff

SourceIsLocal flag to indicate whether local operation

Description

Returns the local tcp port from *skb*.

Name

function::ipmib_tcp_remote_port — Get the remote tcp port

Synopsis

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

Arguments

skb pointer to a struct sk_buff

SourceIsLocal flag to indicate whether local operation

Description

Returns the remote tcp port from *skb*.

Name

function::linuxmib_filter_key — Default filter function for linuxmib.* probes

Synopsis

```
linuxmib_filter_key:long(sk:long,op:long)
```

Arguments

sk pointer to the struct sock

op value to be counted if *sk* passes the filter

Description

This function is a default filter function. The user can replace this function with their own. The user-supplied filter function returns an index key based on the values in *sk*. A return value of 0 means this particular *sk* should be not be counted.

Name

function::tcpmib_filter_key — Default filter function for tcpmib.* probes

Synopsis

```
tcpmib_filter_key:long(sk:long,op:long)
```

Arguments

sk pointer to the struct sock being acted on

op value to be counted if *sk* passes the filter

Description

This function is a default filter function. The user can replace this function with their own. The user-supplied filter function returns an index key based on the values in *sk*. A return value of 0 means this particular *sk* should be not be counted.

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_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_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_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::ipmib.ForwDatagrams — Count forwarded packet

Synopsis

```
ipmib.ForwDatagrams
```

Values

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

skb pointer to the struct `sk_buff` being acted on

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 `IPSTATS_MIB_OUTFORWDATAGRAMS`)

Name

probe::ipmib.FragFails — Count datagram fragmented unsuccessfully

Synopsis

`ipmib.FragFails`

Values

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

skb pointer to the struct `sk_buff` being acted on

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

probe::ipmib.FragOKs — Count datagram fragmented successfully

Synopsis

ipmib.FragOKs

Values

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

skb pointer to the struct sk_buff being acted on

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 IPSTATS_MIB_FRAGOKS)

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 IPSTATS_MIB_INADDRRERORS)

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.InNoRoutes — Count an arriving packet with no matching socket

Synopsis

```
ipmib.InNoRoutes
```

Values

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

skb pointer to the struct `sk_buff` being acted on

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 IPSTATS_MIB_INNOROUTES)

Name

probe::ipmib.InReceives — Count an arriving packet

Synopsis

ipmib.InReceives

Values

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

skb pointer to the struct sk_buff being acted on

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 IPSTATS_MIB_INRECEIVES)

Name

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

Synopsis

```
ipmib.InUnknownProtos
```

Values

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

skb pointer to the struct sk_buff being acted on

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.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 IPSTATS_MIB_OUTREQUESTS)

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 IPSTATS_MIB_REASMREQDS)

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 counted in the global *ReasmTimeout* (equivalent to SNMP's MIB IPSTATS_MIB_REASMTIMEOUT)

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 is counted in the global *DelayedACKs* (equivalent to SNMP's MIB LINUX_MIB_DELAYEDACKS)

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 is counted in the global *ListenDrops* (equivalent to SNMP's MIB LINUX_MIB_LISTENDROPS)

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.TCPMemoryPressures — Count of times memory pressure was used

Synopsis

linuxmib.TCPMemoryPressures

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 *TCPMemoryPressures* (equivalent to SNMP's MIB LINUX_MIB_TCPMEMORYPRESSURES)

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 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

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

sk pointer to the struct sock being acted on

Description

The packet pointed to by *skb* is filtered by the function `tcpmib_filter_key`. If the packet passes the filter is 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

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 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

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

skb pointer to the struct sock being acted on

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

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` (or `ipmib_filter_key` for tcp v4). If the packet passes the filter is 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

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

skb pointer to the struct sock being acted on

Description

The packet pointed to by *skb* is filtered by the function `tcpmib_filter_key`. If the packet passes the filter is 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

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

skb pointer to the struct sock being acted on

Description

The packet pointed to by *skb* is filtered by the function `tcpmib_filter_key`. If the packet passes the filter is 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

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

sk pointer to the struct sock being acted on

Description

The packet pointed to by *skb* is filtered by the function `tcpmib_filter_key`. If the packet passes the filter is 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)

Chapter 16. Kernel Process Tapset

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

Name

function::get_loadavg_index — Get the load average for a specified interval

Synopsis

```
get_loadavg_index:long(indx:long)
```

Arguments

indx The load average interval to capture.

Description

This function returns the load average at a specified interval. The three load average values 1, 5 and 15 minute average corresponds to indexes 0, 1 and 2 of the avenrun array - see linux/sched.h. Please note that the truncated-integer portion of the load average is returned. If the specified index is out-of-bounds, then an error message and exception is thrown.

Name

function::sprint_loadavg — Report a pretty-printed load average

Synopsis

```
sprint_loadavg:string()
```

Arguments

None

Description

Returns the a string with three decimal numbers in the usual format for 1-, 5- and 15-minute load averages.

Name

function::target_set_pid — Does pid descend from target process?

Synopsis

```
target_set_pid(pid:)
```

Arguments

pid The pid of the process to query

Description

This function returns whether the given process-id is within the “target set”, that is whether it is a descendant of the top-level target process.

Name

function::target_set_report — Print a report about the target set

Synopsis

```
target_set_report()
```

Arguments

None

Description

This function prints a report about the processes in the target set, and their ancestry.

Name

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

Synopsis

`kprocess.create`

Values

new_tid The TID of the newly created task

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.exec — Attempt to exec to a new program

Synopsis

`kprocess.exec`

Values

argstr A string containing the filename followed by the arguments to pass, excluding 0th arg (SystemTap v2.5+)

name Name of the system call (“execve”) (SystemTap v2.5+)

args The arguments to pass to the new executable, including the 0th arg (SystemTap v2.5+)

filename The path to the new executable

Context

The caller of `exec`.

Description

Fires whenever a process attempts to `exec` to a new program. Aliased to the `syscall.execve` probe in SystemTap v2.5+.

Name

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

Synopsis

```
kprocess.exec_complete
```

Values

retstr A string representation of *errno* (SystemTap v2.5+)

errno The error number resulting from the *exec*

name Name of the system call (“execve”) (SystemTap v2.5+)

success A boolean indicating whether the *exec* was successful

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. Aliased to the *syscall.execve.return* probe in SystemTap v2.5+.

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

released_pid PID of the process being released

pid Same as *released_pid* for compatibility (deprecated)

released_tid TID of the task 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.

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.

Chapter 17. Signal Tapset

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

Name

function::get_sa_flags — Returns the numeric value of sa_flags

Synopsis

```
get_sa_flags:long(act:long)
```

Arguments

act address of the sigaction to query.

Name

function::get_sa_handler — Returns the numeric value of sa_handler

Synopsis

```
get_sa_handler:long(act:long)
```

Arguments

act address of the sigaction to query.

Name

function::is_sig_blocked — Returns 1 if the signal is currently blocked, or 0 if it is not

Synopsis

```
is_sig_blocked:long(task:long,sig:long)
```

Arguments

task address of the task_struct to query.

sig the signal number to test.

Name

function::sa_flags_str — Returns the string representation of sa_flags

Synopsis

```
sa_flags_str:string(sa_flags:long)
```

Arguments

sa_flags the set of flags to convert to string.

Name

function::sa_handler_str — Returns the string representation of an sa_handler

Synopsis

```
sa_handler_str(handler:)
```

Arguments

handler the sa_handler to convert to string.

Description

Returns the string representation of an sa_handler. If it is not SIG_DFL, SIG_IGN or SIG_ERR, it will return the address of the handler.

Name

function::signal_str — Returns the string representation of a signal number

Synopsis

```
signal_str(num:)
```

Arguments

num the signal number to convert to string.

Name

function::sigset_mask_str — Returns the string representation of a sigset

Synopsis

```
sigset_mask_str:string(mask:long)
```

Arguments

mask the sigset to convert to string.

Name

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

Synopsis

```
signal.check_ignored
```

Values

pid_name Name of the process receiving the signal

sig The number of the signal

sig_name A string representation of the signal

sig_pid 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

name Name of the probe point

retstr Return value as a string

Name

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

Synopsis

`signal.checkperm`

Values

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

Name

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

Synopsis

```
signal.checkperm.return
```

Values

retstr Return value as a string

name Name of the probe point

Name

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

Synopsis

```
signal.do_action
```

Values

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

Name

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

Synopsis

```
signal.do_action.return
```

Values

retstr Return value as a string

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

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_pid</i>	The PID of the process receiving the signal
<i>sig</i>	The number of the signal
<i>pid_name</i>	Name of the process receiving the signal

Name

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

Synopsis

```
signal.force_segv.return
```

Values

name Name of the probe point

retstr Return value as a string

Name

probe::signal.handle — Signal handler being invoked

Synopsis

signal.handle

Values

<i>regs</i>	The address of the kernel-mode stack area (deprecated in SystemTap 2.1)
<i>name</i>	Name of the probe point
<i>sig_name</i>	A string representation of the signal
<i>sig_code</i>	The si_code value of the siginfo signal
<i>sinfo</i>	The address of the siginfo table
<i>ka_addr</i>	The address of the k_sigaction table associated with the signal
<i>sig</i>	The signal number that invoked the signal handler
<i>sig_mode</i>	Indicates whether the signal was a user-mode or kernel-mode signal
<i>oldset_addr</i>	The address of the bitmask array of blocked signals (deprecated in SystemTap 2.1)

Name

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

Synopsis

```
signal.handle.return
```

Values

name Name of the probe point

retstr Return value as a string

Description

(deprecated in SystemTap 2.1)

Name

probe::signal.pending — Examining pending signal

Synopsis

`signal.pending`

Values

name Name of the probe point

sigset_size The size of the user-space signal set

sigset_add 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

name Name of the probe point

retstr Return value as a string

Name

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

Synopsis

signal.procmask

Values

<i>sigset_addr</i>	The address of the signal set (sigset_t) to be implemented
<i>name</i>	Name of the probe point
<i>oldsigset_addr</i>	The old address of the signal set (sigset_t)
<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>sigset</i>	The actual value to be set for sigset_t (correct?)

Name

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

Synopsis

```
signal.procmask.return
```

Values

name Name of the probe point

retstr Return value as a string

Name

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

Synopsis

`signal.send`

Values

<i>pid_name</i>	The name of the signal recipient
<i>shared</i>	Indicates whether the signal is shared by the thread group
<i>si_code</i>	Indicates the signal type
<i>send2queue</i>	Indicates whether the signal is sent to an existing sigqueue (deprecated in SystemTap 2.1)
<i>sig</i>	The number of the signal
<i>sig_name</i>	A string representation of the signal
<i>sinfo</i>	The address of siginfo struct
<i>sig_pid</i>	The PID of the process receiving the signal
<i>name</i>	The name of the function used to send out the signal
<i>task</i>	A task handle to the signal recipient

Context

The signal's sender.

Name

probe::signal.send.return — Signal being sent to a process completed (deprecated in SystemTap 2.1)

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>shared</i>	Indicates whether the sent signal is shared by the thread group.
<i>name</i>	The name of the function used to send out the signal

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.send_sig_queue — Queuing a signal to a process

Synopsis

```
signal.send_sig_queue
```

Values

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

Name

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

Synopsis

```
signal.send_sig_queue.return
```

Values

name Name of the probe point

retstr Return value as a string

Name

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

Synopsis

```
signal.sys_tgkill
```

Values

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

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

retstr The return value to either `__group_send_sig_info`,

name Name of the probe point

Name

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

Synopsis

```
signal.sys_tkill
```

Values

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

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.syskill — Sending kill signal to a process

Synopsis

```
signal.syskill
```

Values

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

Name

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

Synopsis

```
signal.syskill.return
```

Values

None

Name

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

Synopsis

```
signal.systkill.return
```

Values

name Name of the probe point

retstr The return value to either `__group_send_sig_info`,

Name

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

Synopsis

`signal.wakeup`

Values

<i>pid_name</i>	Name of the process to wake
<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, TASK_WAKEKILL, and TASK_INTERRUPTIBLE.
<i>sig_pid</i>	The PID of the process to wake

Chapter 18. Errno Tapset

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

Name

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

Synopsis

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

Arguments

err The error number received

Description

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

Name

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

Synopsis

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

Arguments

format Variable to determine return type base value

ret Return value (typically `$return`)

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`.

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::returnval — Possible return value of probed function

Synopsis

```
returnval:long()
```

Arguments

None

Description

Return the value of the register in which function values are typically returned. Can be used in probes where `$return` isn't available. This is only a guess of the actual return value and can be totally wrong. Normally only used in dwarfless probes.

Chapter 19. RLIMIT Tapset

This set of functions is used to handle string which defines resource limits (RLIMIT_*) and returns corresponding number of resource limit. It contains the following functions:

Name

`function::rlimit_from_str` — Symbolic string associated with resource limit code

Synopsis

```
rlimit_from_str:long(lim_str:string)
```

Arguments

lim_str The string representation of limit

Description

This function returns the number associated with the given string, such as 0 for the string `RLIMIT_CPU`, or -1 for an out-of-range value.

Chapter 20. Device Tapset

This set of functions is used to handle kernel and userspace device numbers. It contains the following functions:

Name

function::MAJOR — Extract major device number from a kernel device number (`kdev_t`)

Synopsis

```
MAJOR:long(dev:long)
```

Arguments

dev Kernel device number to query.

Name

function::MINOR — Extract minor device number from a kernel device number (`kdev_t`)

Synopsis

```
MINOR:long(dev:long)
```

Arguments

dev Kernel device number to query.

Name

function::MKDEV — Creates a value that can be compared to a kernel device number (kdev_t)

Synopsis

```
MKDEV:long(major:long,minor:long)
```

Arguments

major Intended major device number.

minor Intended minor device number.

Name

function::usrdev2kerndev — Converts a user-space device number into the format used in the kernel

Synopsis

```
usrdev2kerndev:long(dev:long)
```

Arguments

dev Device number in user-space format.

Chapter 21. 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::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.

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::inode_path — get the path to an inode

Synopsis

```
inode_path:string(inode:long)
```

Arguments

inode Pointer to inode.

Description

Returns the full path associated with the given inode.

Name

function::real_mount — get the 'struct mount' pointer

Synopsis

```
real_mount:long(vfsmnt:long)
```

Arguments

vfsmnt Pointer to 'struct vfsmount'

Description

Returns the 'struct mount' pointer value for a 'struct vfsmount' pointer.

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

task task_struct pointer.

dentry dentry pointer.

vfsmnt vfsmnt pointer.

Description

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

Chapter 22. Logging Tapset

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

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::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::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.

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 (`relayfs`) 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::printk — Send a message to the kernel trace buffer

Synopsis

```
printk(level:long,msg:string)
```

Arguments

level an integer for the severity level (0=KERN_EMERG ... 7=KERN_DEBUG)

msg The formatted message string

Description

Print a line of text to the kernel dmesg/console with the given severity. An implicit end-of-line is added. This function may not be safely called from all kernel probe contexts, so is restricted to guru mode only.

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 (relays) if it is being used. If the last character is not a newline, the one is added.

Chapter 23. Queue Statistics Tapset

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

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::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_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::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_print — Prints 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.

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_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_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 or requests served per microsecond.

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_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_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`).

Chapter 24. 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 25. 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::atomic_long_read — Retrieves an atomic long variable from kernel memory

Synopsis

```
atomic_long_read:long(addr:long)
```

Arguments

addr pointer to atomic long variable

Description

Safely perform the read of an atomic long variable. This will be a NOP on kernels that do not have ATOMIC_LONG_INIT set on the kernel config.

Name

function::atomic_read — Retrieves an atomic variable from kernel memory

Synopsis

```
atomic_read:long(addr:long)
```

Arguments

addr pointer to atomic variable

Description

Safely perform the read of an atomic variable.

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_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_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_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::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_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

addr The kernel address to retrieve the string from

err_msg 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_string2_utf16 — Retrieves UTF-16 string from kernel memory with alternative error string

Synopsis

```
kernel_string2_utf16:string(addr:long,err_msg:string)
```

Arguments

addr The kernel address to retrieve the string from

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

Description

This function returns a null terminated UTF-8 string converted from the UTF-16 string at a given kernel memory address. Reports the given error message on string copy fault or conversion error.

Name

function::kernel_string2_utf32 — Retrieves UTF-32 string from kernel memory with alternative error string

Synopsis

```
kernel_string2_utf32:string(addr:long,err_msg:string)
```

Arguments

addr The kernel address to retrieve the string from

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

Description

This function returns a null terminated UTF-8 string converted from the UTF-32 string at a given kernel memory address. Reports the given error message on string copy fault or conversion error.

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_string_quoted — Retrieves and quotes string from kernel memory

Synopsis

```
kernel_string_quoted:string(addr:long)
```

Arguments

addr the kernel memory address to retrieve the string from

Description

Returns the null terminated C string from a given kernel memory address where any ASCII characters that are not printable are replaced by the corresponding escape sequence in the returned string. Note that the string will be surrounded by double quotes. If the kernel memory data is not accessible at the given address, the address itself is returned as a string, without double quotes.

Name

function::kernel_string_utf16 — Retrieves UTF-16 string from kernel memory

Synopsis

```
kernel_string_utf16:string(addr:long)
```

Arguments

addr The kernel address to retrieve the string from

Description

This function returns a null terminated UTF-8 string converted from the UTF-16 string at a given kernel memory address. Reports an error on string copy fault or conversion error.

Name

function::kernel_string_utf32 — Retrieves UTF-32 string from kernel memory

Synopsis

```
kernel_string_utf32:string(addr:long)
```

Arguments

addr The kernel address to retrieve the string from

Description

This function returns a null terminated UTF-8 string converted from the UTF-32 string at a given kernel memory address. Reports an error on string copy fault or conversion error.

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_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_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_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_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_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_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_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_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 an error 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

addr the user space address to retrieve the string from

err_msg 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_string2_n_warn — Retrieves string from user space with alternative warning string

Synopsis

```
user_string2_n_warn:string(addr:long,n:long,warn_msg:string)
```

Arguments

addr the user space address to retrieve the string from

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

warn_msg the warning message to return when data isn't available

Description

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

Name

function::user_string2_utf16 — Retrieves UTF-16 string from user memory with alternative error string

Synopsis

```
user_string2_utf16:string(addr:long,err_msg:string)
```

Arguments

addr The user address to retrieve the string from

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

Description

This function returns a null terminated UTF-8 string converted from the UTF-16 string at a given user memory address. Reports the given error message on string copy fault or conversion error.

Name

function::user_string2_utf32 — Retrieves UTF-32 string from user memory with alternative error string

Synopsis

```
user_string2_utf32:string(addr:long,err_msg:string)
```

Arguments

addr The user address to retrieve the string from

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

Description

This function returns a null terminated UTF-8 string converted from the UTF-32 string at a given user memory address. Reports the given error message on string copy fault or conversion error.

Name

function::user_string2_warn — Retrieves string from user space with alternative warning string

Synopsis

```
user_string2_warn:string(addr:long,warn_msg:string)
```

Arguments

addr the user space address to retrieve the string from

warn_msg the warning 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 warning message on the rare cases when userspace data is not accessible and warns (but does not abort) about the failure.

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. Reports an error 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

addr the user space address to retrieve the string from

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

err_msg 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_n2_quoted — Retrieves and quotes string from user space

Synopsis

```
user_string_n2_quoted:string(addr:long,inlen:long,outlen:long)
```

Arguments

addr the user space address to retrieve the string from

inlen the maximum length of the string to read (if not null terminated)

outlen the maximum length of the output string

Description

Reads up to *inlen* characters of a C string from the given user space memory address, and returns up to *outlen* characters, where any ASCII characters that are not printable are replaced by the corresponding escape sequence in the returned string. Note that the string will be surrounded by double quotes. On the rare cases when userspace data is not accessible at the given address, the address itself is returned as a string, without double quotes.

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. Note that the string will be surrounded by double quotes. On the rare cases when userspace data is not accessible at the given address, the address itself is returned as a string, without double quotes.

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_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. Note that the string will be surrounded by double quotes. On the rare cases when userspace data is not accessible at the given address, the address itself is returned as a string, without double quotes.

Name

function::user_string_utf16 — Retrieves UTF-16 string from user memory

Synopsis

```
user_string_utf16:string(addr:long)
```

Arguments

addr The user address to retrieve the string from

Description

This function returns a null terminated UTF-8 string converted from the UTF-16 string at a given user memory address. Reports an error on string copy fault or conversion error.

Name

function::user_string_utf32 — Retrieves UTF-32 string from user memory

Synopsis

```
user_string_utf32:string(addr:long)
```

Arguments

addr The user address to retrieve the string from

Description

This function returns a null terminated UTF-8 string converted from the UTF-32 string at a given user memory address. Reports an error on string copy fault or conversion error.

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 "" on the rare cases when userspace data is not accessible and warns (but does not abort) about the failure.

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_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_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.

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_ulong` — Retrieves an unsigned long value stored in user space

Synopsis

```
user_ulong:long(addr:long)
```

Arguments

addr the user space address to retrieve the unsigned long from

Description

Returns the unsigned long value from a given user space address. Returns zero when user space data is not accessible. Note that the size of the unsigned 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_ulong_warn — Retrieves an unsigned long value stored in user space

Synopsis

```
user_ulong_warn:long(addr:long)
```

Arguments

addr the user space address to retrieve the unsigned long from

Description

Returns the unsigned 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 unsigned 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_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.

Chapter 26. 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_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_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_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_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).

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_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).

Chapter 27. Guru tapsets

Functions to deliberately interfere with the system's behavior, in order to inject faults or improve observability. All the functions in this tapset require the use of guru mode (**-g**).

Name

function::mdelay — millisecond delay

Synopsis

```
mdelay(ms:long)
```

Arguments

ms Number of milliseconds to delay.

Description

This function inserts a multi-millisecond busy-delay into a probe handler. It requires guru mode.

Name

function::panic — trigger a panic

Synopsis

```
panic(msg:string)
```

Arguments

msg message to pass to kernel's `panic` function

Description

This function triggers an immediate panic of the running kernel with a user-specified panic message. It requires guru mode.

Name

function::raise — raise a signal in the current thread

Synopsis

```
raise(signo:long)
```

Arguments

signo signal number

Description

This function calls the kernel `send_sig` routine on the current thread, with the given raw unchecked signal number. It may raise an error if `send_sig` failed. It requires guru mode.

Name

function::udelay — microsecond delay

Synopsis

```
udelay(us:long)
```

Arguments

us Number of microseconds to delay.

Description

This function inserts a multi-microsecond busy-delay into a probe handler. It requires guru mode.

Chapter 28. A collection of standard string functions

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

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::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::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

prnt_str the string to search and replace in

srch_str the substring which is used to search in *prnt_str* string

rplc_str the substring which is used to replace *srch_str*

Description

This function returns the given string with substrings replaced.

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. Reports an error if *pos* is out of bounds.

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::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::substr — Returns a substring

Synopsis

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

Arguments

str the string to take a substring from

start starting position of the extracted string (first character is 0)

length length of string to return

Description

Returns the substring of the given string at the given start position with the given length (or smaller if the length of the original string is less than start + length, or length is bigger than MAXSTRINGLEN).

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

input the string to escape

len maximum length of string to return (0 implies MAXSTRINGLEN)

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

Synopsis

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

Arguments

input string to tokenize. If empty, 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-empty, it returns the first token. If the input string is empty, 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 empty when no more tokens are available.

Chapter 29. 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_cursor_hide — Hides the cursor.

Synopsis

```
ansi_cursor_hide()
```

Arguments

None

Description

Sends ansi code for hiding the cursor.

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_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_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_show — Shows the cursor.

Synopsis

```
ansi_cursor_show( )
```

Arguments

None

Description

Sends ansi code for showing the cursor.

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_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_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::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.

Name

function::indent_depth — returns the global nested-depth

Synopsis

```
indent_depth:long(delta:long)
```

Arguments

delta the amount of depth added/removed for each call

Description

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

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. The timestamp is as per provided by the `__indent_timestamp` function, which by default measures microseconds.

Name

`function::thread_indent_depth` — returns the nested-depth of the current task

Synopsis

```
thread_indent_depth:long(delta:long)
```

Arguments

delta the amount of depth added/removed for each call

Description

This function returns an integer equal to the nested function-call depth starting from the outermost initial level. This function is useful for saving space (consumed by whitespace) in traces with long nested function calls. Use this function in a similar fashion to `thread_indent`, i.e., in call-probe, use `thread_indent_depth(1)` and in return-probe, use `thread_indent_depth(-1)`

Chapter 30. 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.cache_add_mod — Adding kernel instrumentation module to cache

Synopsis

```
stap.cache_add_mod
```

Values

dest_path the path the .ko file is going to (incl filename)

source_path 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_nss — Add NSS (Network Security Services) information to cache

Synopsis

```
stap.cache_add_nss
```

Values

source_path the path the .sgn file is coming from (incl filename)

dest_path 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_add_src — Adding C code translation to cache

Synopsis

```
stap.cache_add_src
```

Values

source_path the path the .c file is coming from (incl filename)

dest_path the path the .c file is going to (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_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_get — Found item in stap cache

Synopsis

`stap.cache_get`

Values

module_path the path of the .ko kernel module file

source_path the path of the .c source file

Description

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

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

Synopsis

```
stap.system.spawn
```

Values

pid the pid of the spawned process

ret the return value from `posix_spawn`

Description

Fires just after the call to `posix_spawn`.

Name

probe::stapio.receive_control_message — Received a control message

Synopsis

```
stapio.receive_control_message
```

Values

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

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

len the length (in bytes) of the data blob

Description

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

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

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

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

len the length (in bytes) of the data blob

Description

Fires at the beginning of the send_request function.

Chapter 31. Network File Storage Tapsets

This family of probe points is used to probe network file storage functions and operations.

Name

function::nfsderror — Convert nfsd error number into string

Synopsis

```
nfsderror:string(err:long)
```

Arguments

err errnum

Description

This function returns a string for the error number passed into the function.

Name

probe::nfs.aop.readpage — NFS client synchronously reading a page

Synopsis

```
nfs.aop.readpage
```

Values

page_index offset within mapping, can used a page identifier and position identifier in the page frame

sb_flag super block flags

file file argument

i_size file length in bytes

dev device identifier

rsize read size (in bytes)

size number of pages to be read in this execution

ino inode number

__page the address of page

i_flag file flags

Description

Read the page over, only fires when a previous async read operation failed

Name

probe::nfs.aop.readpages — NFS client reading multiple pages

Synopsis

`nfs.aop.readpages`

Values

nr_pages number of pages attempted to read in this execution

file filp argument

dev device identifier

rsize read size (in bytes)

size number of pages attempted to read in this execution

ino inode number

rpages read size (in pages)

Description

Fires when in readahead way, read several pages once

Name

probe::nfs.aop.release_page — NFS client releasing page

Synopsis

`nfs.aop.release_page`

Values

page_index offset within mapping, can used a page identifier and position identifier in the page frame

dev device identifier

__page the address of page

ino inode number

size release pages

Description

Fires when do a release operation on NFS.

Name

probe::nfs.aop.set_page_dirty — NFS client marking page as dirty

Synopsis

```
nfs.aop.set_page_dirty
```

Values

page_flag page flags

__page the address of page

Description

This probe attaches to the generic `__set_page_dirty_nobuffers` function. Thus, this probe is going to fire on many other file systems in addition to the NFS client.

Name

probe::nfs.aop.write_begin — NFS client begin to write data

Synopsis

```
nfs.aop.write_begin
```

Values

<i>offset</i>	start address of this write operation
<i>dev</i>	device identifier
<i>__page</i>	the address of page
<i>ino</i>	inode number
<i>size</i>	write bytes
<i>page_index</i>	offset within mapping, can used a page identifier and position identifier in the page frame
<i>to</i>	end address of this write operation

Description

Occurs when write operation occurs on nfs. It prepare a page for writing, look for a request corresponding to the page. If there is one, and it belongs to another file, it flush it out before it tries to copy anything into the page. Also do the same if it finds a request from an existing dropped page

Name

probe::nfs.aop.write_end — NFS client complete writing data

Synopsis

`nfs.aop.write_end`

Values

<i>offset</i>	start address of this write operation
<i>dev</i>	device identifier
<i>i_flag</i>	file flags
<i>__page</i>	the address of page
<i>ino</i>	inode number
<i>size</i>	write bytes
<i>sb_flag</i>	super block flags
<i>page_index</i>	offset within mapping, can used a page identifier and position identifier in the page frame
<i>to</i>	end address of this write operation
<i>i_size</i>	file length in bytes

Description

Fires when do a write operation on nfs, often after `prepare_write`

Update and possibly write a cached page of an NFS file.

Name

probe::nfs.aop.writepage — NFS client writing a mapped page to the NFS server

Synopsis

```
nfs.aop.writepage
```

Values

<i>for_reclaim</i>	a flag of writeback_control, indicates if it's invoked from the page allocator
<i>ino</i>	inode number
<i>size</i>	number of pages to be written in this execution
<i>i_flag</i>	file flags
<i>__page</i>	the address of page
<i>dev</i>	device identifier
<i>i_state</i>	inode state flags
<i>wsiz</i>	write size
<i>i_size</i>	file length in bytes
<i>for_kupdate</i>	a flag of writeback_control, indicates if it's a kupdate writeback
<i>sb_flag</i>	super block flags
<i>page_index</i>	offset within mapping, can used a page identifier and position identifier in the page frame

Description

The priority of wb is decided by the flags *for_reclaim* and *for_kupdate*.

Name

probe::nfs.aop.writepages — NFS client writing several dirty pages to the NFS server

Synopsis

```
nfs.aop.writepages
```

Values

<i>dev</i>	device identifier
<i>nr_to_write</i>	number of pages attempted to be written in this execution
<i>for_reclaim</i>	a flag of writeback_control, indicates if it's invoked from the page allocator
<i>ino</i>	inode number
<i>size</i>	number of pages attempted to be written in this execution
<i>for_kupdate</i>	a flag of writeback_control, indicates if it's a kupdate writeback
<i>wsiz</i>	write size
<i>wpages</i>	write size (in pages)

Description

The priority of wb is decided by the flags *for_reclaim* and *for_kupdate*.

Name

probe::nfs.fop.aio_read — NFS client aio_read file operation

Synopsis

`nfs.fop.aio_read`

Values

<i>pos</i>	current position of file
<i>cache_time</i>	when we started read-caching this inode
<i>parent_name</i>	parent dir name
<i>buf</i>	the address of buf in user space
<i>dev</i>	device identifier
<i>attrtimeo</i>	how long the cached information is assumed to be valid. We need to revalidate the cached attrs for this inode if <code>jiffies - read_cache_jiffies > attrtimeo</code> .
<i>file_name</i>	file name
<i>count</i>	read bytes
<i>cache_valid</i>	cache related bit mask flag
<i>ino</i>	inode number

Name

probe::nfs.fop.aio_write — NFS client aio_write file operation

Synopsis

```
nfs.fop.aio_write
```

Values

<i>ino</i>	inode number
<i>count</i>	read bytes
<i>file_name</i>	file name
<i>dev</i>	device identifier
<i>buf</i>	the address of buf in user space
<i>parent_name</i>	parent dir name
<i>pos</i>	offset of the file

Name

probe::nfs.fop.check_flags — NFS client checking flag operation

Synopsis

`nfs.fop.check_flags`

Values

flag file flag

Name

probe::nfs.fop.flush — NFS client flush file operation

Synopsis

`nfs.fop.flush`

Values

mode file mode

ndirty number of dirty page

ino inode number

dev device identifier

Name

probe::nfs.fop.fsync — NFS client fsync operation

Synopsis

`nfs.fop.fsync`

Values

ndirty number of dirty pages

dev device identifier

ino inode number

Name

probe::nfs.fop.llseek — NFS client llseek operation

Synopsis

`nfs.fop.llseek`

Values

whence_str symbolic string representation of the position to seek from

ino inode number

whence the position to seek from

dev device identifier

offset the offset of the file will be repositioned

Name

probe::nfs.fop.lock — NFS client file lock operation

Synopsis

`nfs.fop.lock`

Values

fl_start starting offset of locked region

fl_end ending offset of locked region

dev device identifier

fl_flag lock flags

ino inode number

i_mode file type and access rights

cmd cmd arguments

fl_type lock type

Name

probe::nfs.fop.mmap — NFS client mmap operation

Synopsis

`nfs.fop.mmap`

Values

<i>vm_start</i>	start address within <code>vm_mm</code>
<i>vm_flag</i>	vm flags
<i>buf</i>	the address of <code>buf</code> in user space
<i>cache_time</i>	when we started read-caching this inode
<i>parent_name</i>	parent dir name
<i>vm_end</i>	the first byte after end address within <code>vm_mm</code>
<i>dev</i>	device identifier
<i>attrtimeo</i>	how long the cached information is assumed to be valid. We need to revalidate the cached attrs for this inode if <code>jiffies - read_cache_jiffies > attrtimeo</code> .
<i>file_name</i>	file name
<i>cache_valid</i>	cache related bit mask flag
<i>ino</i>	inode number

Name

probe::nfs.fop.open — NFS client file open operation

Synopsis

`nfs.fop.open`

Values

<i>dev</i>	device identifier
<i>file_name</i>	file name
<i>ino</i>	inode number
<i>flag</i>	file flag
<i>i_size</i>	file length in bytes

Name

probe::nfs.fop.read — NFS client read operation

Synopsis

`nfs.fop.read`

Values

devname block device name

Description

SystemTap uses the `vfs.do_sync_read` probe to implement this probe and as a result will get operations other than the NFS client read operations.

Name

probe::nfs.fop.read_iter — NFS client read_iter file operation

Synopsis

`nfs.fop.read_iter`

Values

<i>pos</i>	current position of file
<i>cache_time</i>	when we started read-caching this inode
<i>parent_name</i>	parent dir name
<i>dev</i>	device identifier
<i>attrtimeo</i>	how long the cached information is assumed to be valid. We need to revalidate the cached attrs for this inode if <code>jiffies - read_cache_jiffies > attrtimeo</code> .
<i>file_name</i>	file name
<i>count</i>	read bytes
<i>ino</i>	inode number
<i>cache_valid</i>	cache related bit mask flag

Name

probe::nfs.fop.release — NFS client release page operation

Synopsis

`nfs.fop.release`

Values

ino inode number

dev device identifier

mode file mode

Name

probe::nfs.fop.sendfile — NFS client send file operation

Synopsis

`nfs.fop.sendfile`

Values

<i>ppos</i>	current position of file
<i>cache_time</i>	when we started read-caching this inode
<i>attrtimeo</i>	how long the cached information is assumed to be valid. We need to revalidate the cached attrs for this inode if <code>jiffies - read_cache_jiffies > attrtimeo</code> .
<i>dev</i>	device identifier
<i>cache_valid</i>	cache related bit mask flag
<i>ino</i>	inode number
<i>count</i>	read bytes

Name

probe::nfs.fop.write — NFS client write operation

Synopsis

`nfs.fop.write`

Values

devname block device name

Description

SystemTap uses the `vfs.do_sync_write` probe to implement this probe and as a result will get operations other than the NFS client write operations.

Name

probe::nfs.fop.write_iter — NFS client write_iter file operation

Synopsis

`nfs.fop.write_iter`

Values

<i>parent_name</i>	parent dir name
<i>pos</i>	offset of the file
<i>file_name</i>	file name
<i>dev</i>	device identifier
<i>ino</i>	inode number
<i>count</i>	read bytes

Name

probe::nfs.proc.commit — NFS client committing data on server

Synopsis

`nfs.proc.commit`

Values

<i>bitmask0</i>	V4 bitmask representing the set of attributes supported on this filesystem
<i>bitmask1</i>	V4 bitmask representing the set of attributes supported on this filesystem
<i>offset</i>	the file offset
<i>prot</i>	transfer protocol
<i>server_ip</i>	IP address of server
<i>size</i>	read bytes in this execution
<i>version</i>	NFS version

Description

All the `nfs.proc.commit` kernel functions were removed in kernel commit 200baa in December 2006, so these probes do not exist on Linux 2.6.21 and newer kernels.

Fires when client writes the buffered data to disk. The buffered data is asynchronously written by client earlier. The commit function works in sync way. This probe point does not exist in NFSv2.

Name

probe::nfs.proc.commit_done — NFS client response to a commit RPC task

Synopsis

`nfs.proc.commit_done`

Values

timestamp V4 timestamp, which is used for lease renewal

status result of last operation

valid `fattr->valid`, indicates which fields are valid

prot transfer protocol

server_ip IP address of server

count number of bytes committed

version NFS version

Description

Fires when a reply to a commit RPC task is received or some commit operation error occur (timeout or socket shutdown).

Name

probe::nfs.proc.commit_setup — NFS client setting up a commit RPC task

Synopsis

```
nfs.proc.commit_setup
```

Values

<i>offset</i>	the file offset
<i>bitmask1</i>	V4 bitmask representing the set of attributes supported on this filesystem
<i>bitmask0</i>	V4 bitmask representing the set of attributes supported on this filesystem
<i>version</i>	NFS version
<i>size</i>	bytes in this commit
<i>count</i>	bytes in this commit
<i>server_ip</i>	IP address of server
<i>prot</i>	transfer protocol

Description

The `commit_setup` function is used to setup a commit RPC task. It is not doing the actual commit operation. It does not exist in NFSv2.

Name

probe::nfs.proc.create — NFS client creating file on server

Synopsis

`nfs.proc.create`

Values

<i>server_ip</i>	IP address of server
<i>prot</i>	transfer protocol
<i>version</i>	NFS version (the function is used for all NFS version)
<i>fh</i>	file handle of parent dir
<i>filename</i>	file name
<i>filelen</i>	length of file name
<i>flag</i>	indicates create mode (only for NFSv3 and NFSv4)

Name

probe::nfs.proc.handle_exception — NFS client handling an NFSv4 exception

Synopsis

```
nfs.proc.handle_exception
```

Values

errorcode indicates the type of error

Description

This is the error handling routine for processes for NFSv4.

Name

probe::nfs.proc.lookup — NFS client opens/searches a file on server

Synopsis

`nfs.proc.lookup`

Values

<i>version</i>	NFS version
<i>bitmask1</i>	V4 bitmask representing the set of attributes supported on this filesystem
<i>filename</i>	the name of file which client opens/searches on server
<i>name_len</i>	the length of file name
<i>server_ip</i>	IP address of server
<i>bitmask0</i>	V4 bitmask representing the set of attributes supported on this filesystem
<i>prot</i>	transfer protocol

Name

probe::nfs.proc.open — NFS client allocates file read/write context information

Synopsis

```
nfs.proc.open
```

Values

<i>flag</i>	file flag
<i>version</i>	NFS version (the function is used for all NFS version)
<i>mode</i>	file mode
<i>filename</i>	file name
<i>server_ip</i>	IP address of server
<i>prot</i>	transfer protocol

Description

Allocate file read/write context information

Name

probe::nfs.proc.read — NFS client synchronously reads file from server

Synopsis

`nfs.proc.read`

Values

<i>prot</i>	transfer protocol
<i>server_ip</i>	IP address of server
<i>count</i>	read bytes in this execution
<i>flags</i>	used to set task->tk_flags in rpc_init_task function
<i>offset</i>	the file offset
<i>version</i>	NFS version

Description

All the `nfs.proc.read` kernel functions were removed in kernel commit 8e0969 in December 2006, so these probes do not exist on Linux 2.6.21 and newer kernels.

Name

probe::nfs.proc.read_done — NFS client response to a read RPC task

Synopsis

`nfs.proc.read_done`

Values

<i>count</i>	number of bytes read
<i>server_ip</i>	IP address of server
<i>prot</i>	transfer protocol
<i>version</i>	NFS version
<i>status</i>	result of last operation
<i>timestamp</i>	V4 timestamp, which is used for lease renewal

Description

Fires when a reply to a read RPC task is received or some read error occurs (timeout or socket shutdown).

Name

probe::nfs.proc.read_setup — NFS client setting up a read RPC task

Synopsis

```
nfs.proc.read_setup
```

Values

<i>offset</i>	the file offset
<i>prot</i>	transfer protocol
<i>count</i>	read bytes in this execution
<i>server_ip</i>	IP address of server
<i>size</i>	read bytes in this execution
<i>version</i>	NFS version

Description

The `read_setup` function is used to setup a read RPC task. It is not doing the actual read operation.

Name

probe::nfs.proc.release — NFS client releases file read/write context information

Synopsis

```
nfs.proc.release
```

Values

<i>flag</i>	file flag
<i>filename</i>	file name
<i>mode</i>	file mode
<i>version</i>	NFS version (the function is used for all NFS version)
<i>prot</i>	transfer protocol
<i>server_ip</i>	IP address of server

Description

Release file read/write context information

Name

probe::nfs.proc.remove — NFS client removes a file on server

Synopsis

`nfs.proc.remove`

Values

<i>filelen</i>	length of file name
<i>filename</i>	file name
<i>fh</i>	file handle of parent dir
<i>version</i>	NFS version (the function is used for all NFS version)
<i>prot</i>	transfer protocol
<i>server_ip</i>	IP address of server

Name

probe::nfs.proc.rename — NFS client renames a file on server

Synopsis

`nfs.proc.rename`

Values

<i>prot</i>	transfer protocol
<i>server_ip</i>	IP address of server
<i>new_fh</i>	file handle of new parent dir
<i>version</i>	NFS version (the function is used for all NFS version)
<i>old_name</i>	old file name
<i>old_filelen</i>	length of old file name
<i>new_filelen</i>	length of new file name
<i>old_fh</i>	file handle of old parent dir
<i>new_name</i>	new file name

Name

probe::nfs.proc.rename_done — NFS client response to a rename RPC task

Synopsis

`nfs.proc.rename_done`

Values

timestamp V4 timestamp, which is used for lease renewal

status result of last operation

old_fh file handle of old parent dir

prot transfer protocol

new_fh file handle of new parent dir

server_ip IP address of server

version NFS version

Description

Fires when a reply to a rename RPC task is received or some rename error occurs (timeout or socket shutdown).

Name

probe::nfs.proc.rename_setup — NFS client setting up a rename RPC task

Synopsis

```
nfs.proc.rename_setup
```

Values

<i>version</i>	NFS version
<i>fh</i>	file handle of parent dir
<i>server_ip</i>	IP address of server
<i>prot</i>	transfer protocol

Description

The `rename_setup` function is used to setup a rename RPC task. It is not doing the actual rename operation.

Name

probe::nfs.proc.write — NFS client synchronously writes file to server

Synopsis

`nfs.proc.write`

Values

<i>size</i>	read bytes in this execution
<i>version</i>	NFS version
<i>prot</i>	transfer protocol
<i>flags</i>	used to set task->tk_flags in rpc_init_task function
<i>server_ip</i>	IP address of server
<i>bitmask1</i>	V4 bitmask representing the set of attributes supported on this filesystem
<i>offset</i>	the file offset
<i>bitmask0</i>	V4 bitmask representing the set of attributes supported on this filesystem

Description

All the `nfs.proc.write` kernel functions were removed in kernel commit 200baa in December 2006, so these probes do not exist on Linux 2.6.21 and newer kernels.

Name

probe::nfs.proc.write_done — NFS client response to a write RPC task

Synopsis

nfs.proc.write_done

Values

<i>count</i>	number of bytes written
<i>server_ip</i>	IP address of server
<i>prot</i>	transfer protocol
<i>version</i>	NFS version
<i>status</i>	result of last operation
<i>timestamp</i>	V4 timestamp, which is used for lease renewal
<i>valid</i>	fattr->valid, indicates which fields are valid

Description

Fires when a reply to a write RPC task is received or some write error occurs (timeout or socket shutdown).

Name

probe::nfs.proc.write_setup — NFS client setting up a write RPC task

Synopsis

```
nfs.proc.write_setup
```

Values

<i>version</i>	NFS version
<i>size</i>	bytes written in this execution
<i>server_ip</i>	IP address of server
<i>count</i>	bytes written in this execution
<i>how</i>	used to set args.stable. The stable value could be: NFS_UNSTABLE,NFS_DATA_SYNC,NFS_FILE_SYNC (in nfs.proc3.write_setup and nfs.proc4.write_setup)
<i>prot</i>	transfer protocol
<i>offset</i>	the file offset
<i>bitmask1</i>	V4 bitmask representing the set of attributes supported on this filesystem
<i>bitmask0</i>	V4 bitmask representing the set of attributes supported on this filesystem

Description

The write_setup function is used to setup a write RPC task. It is not doing the actual write operation.

Name

probe::nfsd.close — NFS server closing a file for client

Synopsis

`nfsd.close`

Values

filename file name

Name

probe::nfsd.commit — NFS server committing all pending writes to stable storage

Synopsis

`nfsd.commit`

Values

<i>count</i>	read bytes
<i>fh</i>	file handle (the first part is the length of the file handle)
<i>size</i>	read bytes
<i>offset</i>	the offset of file
<i>flag</i>	indicates whether this execution is a sync operation
<i>client_ip</i>	the ip address of client

Name

probe::nfsd.create — NFS server creating a file(regular,dir,device,fifo) for client

Synopsis

`nfsd.create`

Values

iap_valid Attribute flags

type file type(regular,dir,device,fifo ...)

fh file handle (the first part is the length of the file handle)

filename file name

iap_mode file access mode

client_ip the ip address of client

filelen the length of file name

Description

Sometimes nfsd will call `nfsd_create_v3` instead of this this probe point.

Name

probe::nfsd.createv3 — NFS server creating a regular file or set file attributes for client

Synopsis

`nfsd.createv3`

Values

<i>iap_mode</i>	file access mode
<i>fh</i>	file handle (the first part is the length of the file handle)
<i>filename</i>	file name
<i>iap_valid</i>	Attribute flags
<i>filelen</i>	the length of file name
<i>truncp</i>	trunc arguments, indicates if the file should be truncate
<i>verifier</i>	file attributes (atime, mtime, mode). It's used to reset file attributes for CREATE_EXCLUSIVE
<i>createmode</i>	create mode. The possible values could be: NFS3_CREATE_EXCLUSIVE, NFS3_CREATE_UNCHECKED, or NFS3_CREATE_GUARDED
<i>client_ip</i>	the ip address of client

Description

This probe point is only called by `nfsd3_proc_create` and `nfsd4_open` when `op_claim_type` is `NFS4_OPEN_CLAIM_NULL`.

Name

probe::nfsd.dispatch — NFS server receives an operation from client

Synopsis

nfsd.dispatch

Values

<i>prog</i>	program number
<i>xid</i>	transmission id
<i>proc</i>	procedure number
<i>version</i>	nfs version
<i>proto</i>	transfer protocol
<i>client_ip</i>	the ip address of client

Name

probe::nfsd.lookup — NFS server opening or searching file for a file for client

Synopsis

`nfsd.lookup`

Values

filelen the length of file name

client_ip the ip address of client

filename file name

fh file handle of parent dir(the first part is the length of the file handle)

Name

probe::nfsd.open — NFS server opening a file for client

Synopsis

`nfsd.open`

Values

client_ip the ip address of client

access indicates the type of open (read/write/commit/readdir...)

fh file handle (the first part is the length of the file handle)

type type of file (regular file or dir)

Name

probe::nfsd.proc.commit — NFS server performing a commit operation for client

Synopsis

`nfsd.proc.commit`

Values

<i>client_ip</i>	the ip address of client
<i>uid</i>	requester's user id
<i>offset</i>	the offset of file
<i>size</i>	read bytes
<i>gid</i>	requester's group id
<i>version</i>	nfs version
<i>fh</i>	file handle (the first part is the length of the file handle)
<i>proto</i>	transfer protocol
<i>count</i>	read bytes

Name

probe::nfsd.proc.create — NFS server creating a file for client

Synopsis

`nfsd.proc.create`

Values

<i>filename</i>	file name
<i>proto</i>	transfer protocol
<i>fh</i>	file handle (the first part is the length of the file handle)
<i>client_ip</i>	the ip address of client
<i>uid</i>	requester's user id
<i>gid</i>	requester's group id
<i>filelen</i>	length of file name
<i>version</i>	nfs version

Name

probe::nfsd.proc.lookup — NFS server opening or searching for a file for client

Synopsis

`nfsd.proc.lookup`

Values

<i>filename</i>	file name
<i>fh</i>	file handle of parent dir (the first part is the length of the file handle)
<i>proto</i>	transfer protocol
<i>version</i>	nfs version
<i>filelen</i>	the length of file name
<i>client_ip</i>	the ip address of client
<i>uid</i>	requester's user id
<i>gid</i>	requester's group id

Name

probe::nfsd.proc.read — NFS server reading file for client

Synopsis

`nfsd.proc.read`

Values

<i>gid</i>	requester's group id
<i>size</i>	read bytes
<i>offset</i>	the offset of file
<i>uid</i>	requester's user id
<i>client_ip</i>	the ip address of client
<i>version</i>	nfs version
<i>vlen</i>	read blocks
<i>fh</i>	file handle (the first part is the length of the file handle)
<i>vec</i>	struct kvec, includes buf address in kernel address and length of each buffer
<i>proto</i>	transfer protocol
<i>count</i>	read bytes

Name

probe::nfsd.proc.remove — NFS server removing a file for client

Synopsis

`nfsd.proc.remove`

Values

<i>uid</i>	requester's user id
<i>client_ip</i>	the ip address of client
<i>gid</i>	requester's group id
<i>version</i>	nfs version
<i>filelen</i>	length of file name
<i>filename</i>	file name
<i>fh</i>	file handle (the first part is the length of the file handle)
<i>proto</i>	transfer protocol

Name

probe::nfsd.proc.rename — NFS Server renaming a file for client

Synopsis

`nfsd.proc.rename`

Values

<i>fh</i>	file handler of old path
<i>filename</i>	old file name
<i>tfh</i>	file handler of new path
<i>tlen</i>	length of new file name
<i>gid</i>	requester's group id
<i>client_ip</i>	the ip address of client
<i>uid</i>	requester's user id
<i>flen</i>	length of old file name
<i>tname</i>	new file name

Name

probe::nfsd.proc.write — NFS server writing data to file for client

Synopsis

`nfsd.proc.write`

Values

<i>vlen</i>	read blocks
<i>version</i>	nfs version
<i>size</i>	read bytes
<i>gid</i>	requester's group id
<i>client_ip</i>	the ip address of client
<i>uid</i>	requester's user id
<i>offset</i>	the offset of file
<i>count</i>	read bytes
<i>stable</i>	argp->stable
<i>vec</i>	struct kvec, includes buf address in kernel address and length of each buffer
<i>proto</i>	transfer protocol
<i>fh</i>	file handle (the first part is the length of the file handle)

Name

probe::nfsd.read — NFS server reading data from a file for client

Synopsis

`nfsd.read`

Values

<i>vlen</i>	read blocks
<i>size</i>	read bytes
<i>client_ip</i>	the ip address of client
<i>offset</i>	the offset of file
<i>file</i>	argument file, indicates if the file has been opened.
<i>count</i>	read bytes
<i>fh</i>	file handle (the first part is the length of the file handle)
<i>vec</i>	struct kvec, includes buf address in kernel address and length of each buffer

Name

probe::nfsd.rename — NFS server renaming a file for client

Synopsis

`nfsd.rename`

Values

<i>filename</i>	old file name
<i>fh</i>	file handler of old path
<i>tfh</i>	file handler of new path
<i>tlen</i>	length of new file name
<i>flen</i>	length of old file name
<i>tname</i>	new file name
<i>client_ip</i>	the ip address of client

Name

probe::nfsd.unlink — NFS server removing a file or a directory for client

Synopsis

`nfsd.unlink`

Values

client_ip the ip address of client

filelen the length of file name

type file type (file or dir)

fh file handle (the first part is the length of the file handle)

filename file name

Name

probe::nfsd.write — NFS server writing data to a file for client

Synopsis

`nfsd.write`

Values

<i>count</i>	read bytes
<i>fh</i>	file handle (the first part is the length of the file handle)
<i>vec</i>	struct kvec, includes buf address in kernel address and length of each buffer
<i>vlen</i>	read blocks
<i>offset</i>	the offset of file
<i>client_ip</i>	the ip address of client
<i>size</i>	read bytes
<i>file</i>	argument file, indicates if the file has been opened.

Chapter 32. Speculation

This family of functions provides the ability to speculative record information and then at a later point in the SystemTap script either commit the information or discard it.

Name

`function::commit` — Write out all output related to a speculation buffer

Synopsis

```
commit(id:long)
```

Arguments

id of the buffer to store the information in

Description

Output all the output for *id* in the order that it was entered into the speculative buffer by `speculative`.

Name

`function::discard` — Discard all output related to a speculation buffer

Synopsis

```
discard(id:long)
```

Arguments

id of the buffer to store the information in

Name

function::speculate — Store a string for possible output later

Synopsis

```
speculate(id:long,output:string)
```

Arguments

id buffer id to store the information in

output string to write out when commit occurs

Description

Add a string to the speculaive buffer for id.

Name

`function::speculation` — Allocate a new id for speculative output

Synopsis

```
speculation:long()
```

Arguments

None

Description

The `speculation` function is called when a new speculation buffer is needed. It returns an id for the speculative output. There can be multiple threads being speculated on concurrently. This id is used by other speculation functions to keep the threads separate.

Chapter 33. JSON Tapset

This family of probe points, functions, and macros is used to output data in JSON format. It contains the following probe points, functions, and macros:

Name

function::json_add_array — Add an array

Synopsis

```
json_add_array:long(name:string,description:string)
```

Arguments

name The name of the array.

description Array description. An empty string can be used.

Description

This function adds a array, setting up everything needed. Arrays contain other metrics, added with `json_add_array_numeric_metric` or `json_add_array_string_metric`.

Name

function::json_add_array_numeric_metric — Add a numeric metric to an array

Synopsis

```
json_add_array_numeric_metric:long(array_name:string,metric_name:string,metric_desc:
```

Arguments

array_name The name of the array the numeric metric should be added to.

metric_name The name of the numeric metric.

metric_description Metric description. An empty string can be used.

metric_units Metic units. An empty string can be used.

Description

This function adds a numeric metric to an array, setting up everything needed.

Name

function::json_add_array_string_metric — Add a string metric to an array

Synopsis

```
json_add_array_string_metric:long(array_name:string,metric_name:string,metric_descr
```

Arguments

array_name The name of the array the string metric should be added to.

metric_name The name of the string metric.

metric_description Metric description. An empty string can be used.

Description

This function adds a string metric to an array, setting up everything needed.

Name

function::json_add_numeric_metric — Add a numeric metric

Synopsis

```
json_add_numeric_metric:long(name:string,description:string,units:string)
```

Arguments

name The name of the numeric metric.

description Metric description. An empty string can be used.

units Metic units. An empty string can be used.

Description

This function adds a numeric metric, setting up everything needed.

Name

function::json_add_string_metric — Add a string metric

Synopsis

```
json_add_string_metric:long(name:string,description:string)
```

Arguments

name The name of the string metric.

description Metric description. An empty string can be used.

Description

This function adds a string metric, setting up everything needed.

Name

function::json_set_prefix — Set the metric prefix.

Synopsis

```
json_set_prefix:long(prefix:string)
```

Arguments

prefix The prefix name to be used.

Description

This function sets the “prefix”, which is the name of the base of the metric hierarchy. Calling this function is optional, by default the name of the systemtap module is used.

Name

macro::json_output_array_numeric_value — Output a numeric value for metric in an array.

Synopsis

```
@json_output_array_numeric_value(array_name,array_index,metric_name,value)
```

Arguments

array_name The name of the array.

array_index The array index (as a string) indicating where to store the numeric value.

metric_name The name of the numeric metric.

value The numeric value to output.

Description

The `json_output_array_numeric_value` macro is designed to be called from the 'json_data' probe in the user's script to output a metric's numeric value that is in an array. This metric should have been added with `json_add_array_numeric_metric`.

Name

macro::json_output_array_string_value — Output a string value for metric in an array.

Synopsis

```
@json_output_array_string_value(array_name,array_index,metric_name,value)
```

Arguments

array_name The name of the array.

array_index The array index (as a string) indicating where to store the string value.

metric_name The name of the string metric.

value The string value to output.

Description

The `json_output_array_string_value` macro is designed to be called from the 'json_data' probe in the user's script to output a metric's string value that is in an array. This metric should have been added with `json_add_array_string_metric`.

Name

macro::json_output_data_end — End the json output.

Synopsis

```
@json_output_data_end()
```

Arguments

None

Description

The `json_output_data_end` macro is designed to be called from the 'json_data' probe from the user's script. It marks the end of the JSON output.

Name

macro::json_output_data_start — Start the json output.

Synopsis

```
@json_output_data_start()
```

Arguments

None

Description

The json_output_data_start macro is designed to be called from the 'json_data' probe from the user's script. It marks the start of the JSON output.

Name

macro::json_output_numeric_value — Output a numeric value.

Synopsis

```
@json_output_numeric_value(name,value)
```

Arguments

name The name of the numeric metric.

value The numeric value to output.

Description

The `json_output_numeric_value` macro is designed to be called from the 'json_data' probe in the user's script to output a metric's numeric value. This metric should have been added with `json_add_numeric_metric`.

Name

macro::json_output_string_value — Output a string value.

Synopsis

```
@json_output_string_value(name,value)
```

Arguments

name The name of the string metric.

value The string value to output.

Description

The `json_output_string_value` macro is designed to be called from the 'json_data' probe in the user's script to output a metric's string value. This metric should have been added with `json_add_string_metric`.

Name

probe::json_data — Fires whenever JSON data is wanted by a reader.

Synopsis

json_data

Values

None

Context

This probe fires when the JSON data is about to be read. This probe must gather up data and then call the following macros to output the data in JSON format. First, *json_output_data_start()* must be called. That call is followed by one or more of the following (one call for each data item): *json_output_string_value()*, *json_output_numeric_value()*, *json_output_array_string_value()*, and *json_output_array_numeric_value()*. Finally *json_output_data_end()* must be called.