

SystemTap Tapset Reference Manual

SystemTap

SystemTap Tapset Reference Manual

by SystemTap

Copyright © 2008-2015 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::current_exe_file	14
function::egid	15
function::env_var	16
function::euid	17
function::execname	18
function::fastcall	19
function::gid	20
function::int_arg	21
function::is_myproc	22
function::is_return	23
function::long_arg	24
function::longlong_arg	25
function::modname	26
function::module_name	27
function::module_size	28
function::ns_egid	29
function::ns_euid	30
function::ns_gid	31
function::ns_pgrp	32
function::ns_pid	33
function::ns_ppid	34
function::ns_sid	35
function::ns_tid	36
function::ns_uid	37
function::pexecname	38
function::pgrp	39
function::pid	40
function::pid2execname	41
function::pid2task	42
function::pn	43
function::pnlabel	44
function::pointer_arg	45
function::pp	46
function::ppfunc	47
function::ppid	48
function::print_backtrace	49
function::print_regs	50
function::print_stack	51

function::print_syms	52
function::print_ubacktrace	53
function::print_ubacktrace_brief	54
function::print_ustack	55
function::print_usyms	56
function::probe_type	57
function::probecfunc	58
function::probemod	59
function::pstrace	60
function::register	61
function::registers_valid	62
function::regparm	63
function::remote_id	64
function::remote_uri	65
function::s32_arg	66
function::s64_arg	67
function::sid	68
function::sprint_backtrace	69
function::sprint_stack	70
function::sprint_syms	71
function::sprint_ubacktrace	72
function::sprint_ustack	73
function::sprint_usyms	74
function::stack	75
function::stack_size	76
function::stack_unused	77
function::stack_used	78
function::stp_pid	79
function::symdata	80
function::symfile	81
function::symfileline	82
function::symline	83
function::symname	84
function::target	85
function::task_ancestry	86
function::task_backtrace	87
function::task_cpu	88
function::task_current	89
function::task_cwd_path	90
function::task_egid	91
function::task_euid	92
function::task_exe_file	93
function::task_execname	94
function::task_fd_lookup	95
function::task_gid	96
function::task_max_file_handles	97
function::task_nice	98
function::task_ns_egid	99
function::task_ns_euid	100
function::task_ns_gid	101
function::task_ns_pid	102
function::task_ns_tid	103
function::task_ns_uid	104
function::task_open_file_handles	105

function::task_parent	106
function::task_pid	107
function::task_prio	108
function::task_state	109
function::task_tid	110
function::task_uid	111
function::tid	112
function::u32_arg	113
function::u64_arg	114
function::u_register	115
function::uaddr	116
function::ubacktrace	117
function::ucallers	118
function::uid	119
function::uint_arg	120
function::ulong_arg	121
function::ulonglong_arg	122
function::umodname	123
function::user_mode	124
function::ustack	125
function::usymdata	126
function::usymfile	127
function::usymfileline	128
function::usymline	129
function::usymname	130
3. Timestamp Functions	131
function::HZ	132
function::cpu_clock_ms	133
function::cpu_clock_ns	134
function::cpu_clock_s	135
function::cpu_clock_us	136
function::delete_stopwatch	137
function::get_cycles	138
function::gettimeofday_ms	139
function::gettimeofday_ns	140
function::gettimeofday_s	141
function::gettimeofday_us	142
function::jiffies	143
function::local_clock_ms	144
function::local_clock_ns	145
function::local_clock_s	146
function::local_clock_us	147
function::read_stopwatch_ms	148
function::read_stopwatch_ns	149
function::read_stopwatch_s	150
function::read_stopwatch_us	151
function::start_stopwatch	152
function::stop_stopwatch	153
4. Time utility functions	154
function::ctime	155
function::tz_ctime	156
function::tz_gmtoff	157
function::tz_name	158
5. Shell command functions	159

function::system	160
6. Memory Tapset	161
function::addr_to_node	162
function::bytes_to_string	163
function::mem_page_size	164
function::pages_to_string	165
function::proc_mem_data	166
function::proc_mem_rss	167
function::proc_mem_shr	168
function::proc_mem_size	169
function::proc_mem_string	170
function::proc_mem_txt	171
function::vm_fault_contains	172
probe::vm.brk	173
probe::vm.kfree	174
probe::vm.kmalloc	175
probe::vm.kmalloc_node	176
probe::vm.kmem_cache_alloc	177
probe::vm.kmem_cache_alloc_node	178
probe::vm.kmem_cache_free	179
probe::vm.mmap	180
probe::vm.munmap	181
probe::vm.oom_kill	182
probe::vm.pagefault	183
probe::vm.pagefault.return	184
probe::vm.write_shared	185
probe::vm.write_shared_copy	186
7. Task Time Tapset	187
function::cputime_to_msecs	188
function::cputime_to_string	189
function::cputime_to_usecs	190
function::msecs_to_string	191
function::nsecs_to_string	192
function::task_start_time	193
function::task_stime	194
function::task_time_string	195
function::task_time_string_tid	196
function::task_utime	197
function::usecs_to_string	198
8. Scheduler Tapset	199
probe::scheduler.balance	200
probe::scheduler.cpu_off	201
probe::scheduler.cpu_on	202
probe::scheduler.ctxswitch	203
probe::scheduler.kthread_stop	204
probe::scheduler.kthread_stop.return	205
probe::scheduler.migrate	206
probe::scheduler.process_exit	207
probe::scheduler.process_fork	208
probe::scheduler.process_free	209
probe::scheduler.process_wait	210
probe::scheduler.signal_send	211
probe::scheduler.tick	212
probe::scheduler.wait_task	213

probe::scheduler.wakeup	214
probe::scheduler.wakeup_new	215
9. IO Scheduler and block IO Tapset	216
probe::ioblock.end	217
probe::ioblock.request	218
probe::ioblock_trace.bounce	219
probe::ioblock_trace.end	220
probe::ioblock_trace.request	221
probe::ioscheduler.elv_add_request	222
probe::ioscheduler.elv_add_request.kp	223
probe::ioscheduler.elv_add_request.tp	224
probe::ioscheduler.elv_completed_request	225
probe::ioscheduler.elv_next_request	226
probe::ioscheduler.elv_next_request.return	227
probe::ioscheduler_trace.elv_abort_request	228
probe::ioscheduler_trace.elv_completed_request	229
probe::ioscheduler_trace.elv_issue_request	230
probe::ioscheduler_trace.elv_requeue_request	231
probe::ioscheduler_trace.plugin	232
probe::ioscheduler_trace.unplug_io	233
probe::ioscheduler_trace.unplug_timer	234
10. SCSI Tapset	235
probe::scsi.iocompleted	236
probe::scsi.iodispatching	237
probe::scsi.iodone	238
probe::scsi.ioentry	239
probe::scsi.ioexecute	240
probe::scsi.set_state	241
11. TTY Tapset	242
probe::tty.init	243
probe::tty.ioctl	244
probe::tty.open	245
probe::tty.poll	246
probe::tty.read	247
probe::tty.receive	248
probe::tty.register	249
probe::tty.release	250
probe::tty.resize	251
probe::tty.unregister	252
probe::tty.write	253
12. Interrupt Request (IRQ) Tapset	254
probe::irq_handler.entry	255
probe::irq_handler.exit	256
probe::softirq.entry	257
probe::softirq.exit	258
probe::workqueue.create	259
probe::workqueue.destroy	260
probe::workqueue.execute	261
probe::workqueue.insert	262
13. Networking Tapset	263
function::format_ipaddr	264
function::htonl	265
function::htonll	266
function::htons	267

function::ip_ntop	268
function::ntohl	269
function::ntohll	270
function::ntohs	271
probe::netdev.change_mac	272
probe::netdev.change_mtu	273
probe::netdev.change_rx_flag	274
probe::netdev.close	275
probe::netdev.get_stats	276
probe::netdev.hard_transmit	277
probe::netdev.ioctl	278
probe::netdev.open	279
probe::netdev.receive	280
probe::netdev.register	281
probe::netdev.rx	282
probe::netdev.set_promiscuity	283
probe::netdev.transmit	284
probe::netdev.unregister	285
probe::netfilter.arp.forward	286
probe::netfilter.arp.in	288
probe::netfilter.arp.out	290
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_tgkill	452
probe::signal.sys_tgkill.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::fullpath_struct_file	474
function::fullpath_struct_nameidata	475
function::fullpath_struct_path	476
function::inode_name	477
function::inode_path	478
function::real_mount	479
function::reverse_path_walk	480
function::task_dentry_path	481
22. Logging Tapset	482
function::assert	483
function::error	484
function::exit	485
function::ftrace	486
function::log	487
function::printk	488
function::warn	489
23. Queue Statistics Tapset	490
function::qs_done	491
function::qs_run	492
function::qs_wait	493
function::qsq_blocked	494
function::qsq_print	495
function::qsq_service_time	496

function::qsq_start	497
function::qsq_throughput	498
function::qsq_utilization	499
function::qsq_wait_queue_length	500
function::qsq_wait_time	501
24. Random functions Tapset	502
function::randint	503
25. String and data retrieving functions Tapset	504
function::atomic_long_read	505
function::atomic_read	506
function::kernel_buffer_quoted	507
function::kernel_buffer_quoted_error	508
function::kernel_char	509
function::kernel_int	510
function::kernel_long	511
function::kernel_pointer	512
function::kernel_short	513
function::kernel_string	514
function::kernel_string_n	515
function::kernel_string_quoted	516
function::kernel_string_quoted_utf16	517
function::kernel_string_quoted_utf32	518
function::kernel_string_utf16	519
function::kernel_string_utf32	520
function::user_buffer_quoted	521
function::user_buffer_quoted_error	522
function::user_char	523
function::user_char_error	524
function::user_char_warn	525
function::user_int	526
function::user_int16	527
function::user_int16_error	528
function::user_int32	529
function::user_int32_error	530
function::user_int64	531
function::user_int64_error	532
function::user_int8	533
function::user_int8_error	534
function::user_int_error	535
function::user_int_warn	536
function::user_long	537
function::user_long_error	538
function::user_long_warn	539
function::user_short	540
function::user_short_error	541
function::user_short_warn	542
function::user_string	543
function::user_string_n	544
function::user_string_n_quoted	545
function::user_string_n_warn	546
function::user_string_quoted	547
function::user_string_quoted_utf16	548
function::user_string_quoted_utf32	549
function::user_string_utf16	550

function::user_string_utf32	551
function::user_string_warn	552
function::user_uint16	553
function::user_uint16_error	554
function::user_uint32	555
function::user_uint32_error	556
function::user_uint64	557
function::user_uint64_error	558
function::user_uint8	559
function::user_uint8_error	560
function::user_ulong	561
function::user_ulong_error	562
function::user_ulong_warn	563
function::user_ushort	564
function::user_ushort_error	565
function::user_ushort_warn	566
26. String and data writing functions Tapset	567
function::set_kernel_char	568
function::set_kernel_int	569
function::set_kernel_long	570
function::set_kernel_pointer	571
function::set_kernel_short	572
function::set_kernel_string	573
function::set_kernel_string_n	574
function::set_user_char	575
function::set_user_int	576
function::set_user_long	577
function::set_user_pointer	578
function::set_user_short	579
function::set_user_string	580
function::set_user_string_n	581
27. Guru tapsets	582
function::mdelay	583
function::panic	584
function::raise	585
function::udelay	586
28. A collection of standard string functions	587
function::isdigit	588
function::isinstr	589
function::matched	590
function::matched_str	591
function::ngroups	592
function::str_replace	593
function::string_quoted	594
function::stringat	595
function::strlen	596
function::strpos	597
function::strtol	598
function::substr	599
function::text_str	600
function::text_strn	601
function::tokenize	602
29. Utility functions for using ansi control chars in logs	603
function::ansi_clear_screen	604

function::ansi_cursor_hide	605
function::ansi_cursor_move	606
function::ansi_cursor_restore	607
function::ansi_cursor_save	608
function::ansi_cursor_show	609
function::ansi_new_line	610
function::ansi_reset_color	611
function::ansi_set_color	612
function::indent	613
function::indent_depth	614
function::thread_indent	615
function::thread_indent_depth	616
30. SystemTap Translator Tapset	617
probe::stap.cache_add_mod	618
probe::stap.cache_add_nss	619
probe::stap.cache_add_src	620
probe::stap.cache_clean	621
probe::stap.cache_get	622
probe::stap.pass0	623
probe::stap.pass0.end	624
probe::stap.pass1.end	625
probe::stap.pass1a	626
probe::stap.pass1b	627
probe::stap.pass2	628
probe::stap.pass2.end	629
probe::stap.pass3	630
probe::stap.pass3.end	631
probe::stap.pass4	632
probe::stap.pass4.end	633
probe::stap.pass5	634
probe::stap.pass5.end	635
probe::stap.pass6	636
probe::stap.pass6.end	637
probe::stap.system	638
probe::stap.system.return	639
probe::stap.system.spawn	640
probe::stapio.receive_control_message	641
probe::staprun.insert_module	642
probe::staprun.remove_module	643
probe::staprun.send_control_message	644
31. Network File Storage Tapsets	645
function::nfsderror	646
probe::nfs.aop.readpage	647
probe::nfs.aop.readpages	648
probe::nfs.aop.release_page	649
probe::nfs.aop.set_page_dirty	650
probe::nfs.aop.write_begin	651
probe::nfs.aop.write_end	652
probe::nfs.aop.writepage	653
probe::nfs.aop.writepages	654
probe::nfs.fop.aio_read	655
probe::nfs.fop.aio_write	656
probe::nfs.fop.check_flags	657
probe::nfs.fop.flush	658

probe::nfs.fop.fsync	659
probe::nfs.fop.llseek	660
probe::nfs.fop.lock	661
probe::nfs.fop.mmap	662
probe::nfs.fop.open	663
probe::nfs.fop.read	664
probe::nfs.fop.read_iter	665
probe::nfs.fop.release	666
probe::nfs.fop.sendfile	667
probe::nfs.fop.write	668
probe::nfs.fop.write_iter	669
probe::nfs.proc.commit	670
probe::nfs.proc.commit_done	671
probe::nfs.proc.commit_setup	672
probe::nfs.proc.create	673
probe::nfs.proc.handle_exception	674
probe::nfs.proc.lookup	675
probe::nfs.proc.open	676
probe::nfs.proc.read	677
probe::nfs.proc.read_done	678
probe::nfs.proc.read_setup	679
probe::nfs.proc.release	680
probe::nfs.proc.remove	681
probe::nfs.proc.rename	682
probe::nfs.proc.rename_done	683
probe::nfs.proc.rename_setup	684
probe::nfs.proc.write	685
probe::nfs.proc.write_done	686
probe::nfs.proc.write_setup	687
probe::nfsd.close	688
probe::nfsd.commit	689
probe::nfsd.create	690
probe::nfsd.createv3	691
probe::nfsd.dispatch	692
probe::nfsd.lookup	693
probe::nfsd.open	694
probe::nfsd.proc.commit	695
probe::nfsd.proc.create	696
probe::nfsd.proc.lookup	697
probe::nfsd.proc.read	698
probe::nfsd.proc.remove	699
probe::nfsd.proc.rename	700
probe::nfsd.proc.write	701
probe::nfsd.read	702
probe::nfsd.rename	703
probe::nfsd.unlink	704
probe::nfsd.write	705
32. Speculation	706
function::commit	707
function::discard	708
function::speculate	709
function::speculation	710
33. JSON Tapset	711
function::json_add_array	712

function::json_add_array_numeric_metric	713
function::json_add_array_string_metric	714
function::json_add_numeric_metric	715
function::json_add_string_metric	716
function::json_set_prefix	717
macro::json_output_array_numeric_value	718
macro::json_output_array_string_value	719
macro::json_output_data_end	720
macro::json_output_data_start	721
macro::json_output_numeric_value	722
macro::json_output_string_value	723
probe::json_data	724
34. Output file switching Tapset	725
function::switch_file	726

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.

function::addr

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`.

function::asmlinkage

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.

function::backtrace

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.

function::caller

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)`

function::caller_addr

function::caller_addr — Return caller address

Synopsis

```
caller_addr:long()
```

Arguments

None

Description

This function returns the address of the calling function.

function::callers

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).

function::cmdline_arg

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.

function::cmdline_args

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

Synopsis

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

Arguments

<i>n</i>	First argument to get (zero is normally the program itself)
<i>m</i>	Last argument to get (or minus one for all arguments after n)
<i>delim</i>	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.

function::cmdline_str

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.

function::cpu

function::cpu — Returns the current cpu number

Synopsis

```
cpu:long( )
```

Arguments

None

Description

This function returns the current cpu number.

function::cpuid

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.

function::current_exe_file

function::current_exe_file — get the file struct pointer for the current task's executable file

Synopsis

```
current_exe_file:long()
```

Arguments

None

Description

This function returns the file struct pointer for the current task's executable file. Note that the file struct pointer isn't locked on return. The return value of this function can be passed to `fullpath_struct_file` to get the path from the file struct.

function::egid

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

function::env_var

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.

function::euid

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.

function::execname

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).

function::fastcall

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.

function::gid

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.

function::int_arg

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).

function::is_myproc

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.

function::is_return

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.

function::long_arg

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.

function::longlong_arg

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.

function::modname

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.

function::module_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>.

function::module_size

function::module_size — The module size of the current script

Synopsis

```
module_size:string()
```

Arguments

None

Description

This function returns the sizes of various sections of the stap module.

function::ns_egid

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

function::ns_euid

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.

function::ns_gid

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.

function::ns_pgrp

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.

function::ns_pid

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.

function::ns_ppid

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.

function::ns_sid

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.

function::ns_tid

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.

function::ns_uid

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.

function::pexecname

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.

function::pgrp

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.

function::pid

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

Synopsis

```
pid:long()
```

Arguments

None

Description

This function returns the ID of a target process.

function::pid2execname

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.

function::pid2task

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.

function::pn

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.

function::pnlabel

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.

function::pointer_arg

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.

function::pp

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.

function::ppfunc

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.

function::ppid

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.

function::print_backtrace

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.

function::print_regs

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.

function::print_stack

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.

function::print_syms

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.

function::print_ubacktrace

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.

function::print_ubacktrace_brief

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 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.

function::print_ustack

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.

function::print_usyms

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.

function::probe_type

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).

function::probefunc

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.

function::probemod

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.

function::pstrace

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.

function::register

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.

function::registers_valid

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.

function::regparm

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.

function::remote_id

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.

function::remote_uri

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”.

function::s32_arg

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`.

function::s64_arg

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`.

function::sid

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.

function::sprint_backtrace

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).

function::sprint_stack

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.

function::sprint_syms

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 `symdata`. 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`.

function::sprint_ubacktrace

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.

function::sprint_ustack

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.

function::sprint_usyms

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`.

function::stack

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.

function::stack_size

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.

function::stack_unused

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.

function::stack_used

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.

function::stp_pid

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.

function::symdata

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.

function::symfile

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.

function::symfileline

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.

function::symline

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.

function::symname

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*.

function::target

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

function::task_ancestry

function::task_ancestry — The ancestry of the given task

Synopsis

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

Arguments

<i>task</i>	task_struct pointer
<i>with_time</i>	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”.

function::task_backtrace

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.

function::task_cpu

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.

function::task_current

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.

function::task_cwd_path

function::task_cwd_path — get the path struct pointer for a task's current working directory

Synopsis

```
task_cwd_path:long(task:long)
```

Arguments

task task_struct pointer.

function::task_egid

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.

function::task_euid

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.

function::task_exe_file

function::task_exe_file — get the file struct pointer for a task's executable file

Synopsis

```
task_exe_file:long(task:long)
```

Arguments

task task_struct pointer.

function::task_execname

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.

function::task_fd_lookup

function::task_fd_lookup — get the file struct for a task's fd

Synopsis

```
task_fd_lookup:long(task:long,fd:long)
```

Arguments

task task_struct pointer.

fd file descriptor number.

Description

Returns the file struct pointer for a task's file descriptor.

function::task_gid

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.

function::task_max_file_handles

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.

function::task_nice

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.

function::task_ns_egid

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.

function::task_ns_euid

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.

function::task_ns_gid

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.

function::task_ns_pid

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..

function::task_ns_tid

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.

function::task_ns_uid

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.

function::task_open_file_handles

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.

function::task_parent

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.

function::task_pid

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.

function::task_prio

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.

function::task_state

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).

function::task_tid

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.

function::task_uid

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.

function::tid

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.

function::u32_arg

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`.

function::u64_arg

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`.

function::u_register

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.

function::uaddr

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.

function::ubacktrace

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.

function::ucallers

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.

function::uid

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.

function::uint_arg

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).

function::ulong_arg

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.

function::ulonglong_arg

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`.)

function::umodname

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.

function::user_mode

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.

function::ustack

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.

function::usymdata

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.

function::usymfile

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.

function::usymfileline

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.

function::usymline

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.

function::usymname

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.

function::HZ

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.

function::cpu_clock_ms

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).

function::cpu_clock_ns

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).

function::cpu_clock_s

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).

function::cpu_clock_us

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).

function::delete_stopwatch

function::delete_stopwatch — Remove an existing stopwatch

Synopsis

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

Arguments

name the stopwatch name

Description

Remove stopwatch *name*.

function::get_cycles

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.

function::gettimeofday_ms

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.

function::gettimeofday_ns

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.

function::gettimeofday_s

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.

function::gettimeofday_us

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.

function::jiffies

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.

function::local_clock_ms

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).

function::local_clock_ns

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).

function::local_clock_s

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).

function::local_clock_us

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).

function::read_stopwatch_ms

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.

function::read_stopwatch_ns

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.

function::read_stopwatch_s

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.

function::read_stopwatch_us

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.

function::start_stopwatch

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.

function::stop_stopwatch

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.

function::ctime

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

Synopsis

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

Arguments

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

Description

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

2) “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.

function::tz_ctime

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.

function::tz_gmtoff

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.

function::tz_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.

function::system

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. The runtime may impose a relatively short length limit on the command string. Exceeding it may print a warning.

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:

function::addr_to_node

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.

function::bytes_to_string

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.

function::mem_page_size

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

Synopsis

```
mem_page_size:long()
```

Arguments

None

function::pages_to_string

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`.

function::proc_mem_data

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

Synopsis

- 1) `proc_mem_data:long()`
- 2) `proc_mem_data:long(pid:long)`

Arguments

pid The pid of process to examine

Description

- 1) 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.
- 2) 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.

function::proc_mem_rss

function::proc_mem_rss — Program resident set size in pages

Synopsis

- 1) `proc_mem_rss:long()`
- 2) `proc_mem_rss:long(pid:long)`

Arguments

pid The pid of process to examine

Description

- 1) 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.
- 2) 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.

function::proc_mem_shr

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

Synopsis

- 1) `proc_mem_shr:long()`
- 2) `proc_mem_shr:long(pid:long)`

Arguments

pid The pid of process to examine

Description

- 1) 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.
- 2) 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.

function::proc_mem_size

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

Synopsis

- 1) `proc_mem_size:long()`
- 2) `proc_mem_size:long(pid:long)`

Arguments

pid The pid of process to examine

Description

- 1) 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.
- 2) 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.

function::proc_mem_string

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

Synopsis

- 1) `proc_mem_string:string()`
- 2) `proc_mem_string:string(pid:long)`

Arguments

pid The pid of process to examine

Description

- 1) 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”.
- 2) 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”.

function::proc_mem_txt

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

Synopsis

- 1) `proc_mem_txt:long()`
- 2) `proc_mem_txt:long(pid:long)`

Arguments

pid The pid of process to examine

Description

- 1) 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.
- 2) 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.

function::vm_fault_contains

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

Synopsis

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

Arguments

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

probe::vm.brk

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

Synopsis

vm.brk

Values

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

Context

The process calling brk.

probe::vm.kfree

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

Synopsis

vm.kfree

Values

<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>ptr</i>	pointer to the kmemory allocated which is returned by kmalloc

probe::vm.kmalloc

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

Synopsis

```
vm.kmalloc
```

Values

<i>name</i>	name of the probe point
<i>call_site</i>	address of the kmemory function
<i>caller_function</i>	name of the caller function
<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>ptr</i>	pointer to the kmemory allocated
<i>bytes_req</i>	requested Bytes

probe::vm.kmalloc_node

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

Synopsis

vm.kmalloc_node

Values

<i>name</i>	name of the probe point
<i>call_site</i>	address of the function calling this kmemory function
<i>caller_function</i>	name of the caller function
<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>ptr</i>	pointer to the kmemory allocated

probe::vm.kmem_cache_alloc

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

Synopsis

vm.kmem_cache_alloc

Values

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

probe::vm.kmem_cache_alloc_node

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

Synopsis

vm.kmem_cache_alloc_node

Values

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

probe::vm.kmem_cache_free

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

Synopsis

vm.kmem_cache_free

Values

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

probe::vm.mmap

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

Synopsis

`vm.mmap`

Values

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

Context

The process calling mmap.

probe::vm.munmap

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

Synopsis

`vm.munmap`

Values

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

Context

The process calling munmap.

probe::vm.oom_kill

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.

probe::vm.pagefault

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

Synopsis

`vm.pagefault`

Values

<i>name</i>	name of the probe point
<i>write_access</i>	indicates whether this was a write or read access; 1 indicates a write, while 0 indicates a read
<i>address</i>	the address of the faulting memory access; i.e. the address that caused the page fault

Context

The process which triggered the fault

probe::vm.pagefault.return

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

Synopsis

```
vm.pagefault.return
```

Values

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

probe::vm.write_shared

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

Synopsis

```
vm.write_shared
```

Values

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

Context

The context is the process attempting the write.

Description

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

probe::vm.write_shared_copy

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

Synopsis

```
vm.write_shared_copy
```

Values

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

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.

function::cputime_to_msecs

function::cputime_to_msecs — Translates the given cputime into milliseconds

Synopsis

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

Arguments

cputime Time to convert to milliseconds.

function::cputime_to_string

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)).

function::cputime_to_usecs

function::cputime_to_usecs — Translates the given cputime into microseconds

Synopsis

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

Arguments

cputime Time to convert to microseconds.

function::msecs_to_string

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.

function::nsecs_to_string

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.

function::task_start_time

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.

function::task_stime

function::task_stime — System time of the task

Synopsis

- 1) `task_stime:long()`
- 2) `task_stime:long(tid:long)`

Arguments

tid Thread id of the given task

Description

- 1) 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.
- 2) 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.

function::task_time_string

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”.

function::task_time_string_tid

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”.

function::task_etime

function::task_etime — User time of the task

Synopsis

- 1) task_etime:long()
- 2) task_etime:long(tid:long)

Arguments

tid Thread id of the given task

Description

- 1) 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.
- 2) 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.

function::usecs_to_string

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:

probe::scheduler.balance

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.

probe::scheduler.cpu_off

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

Synopsis

```
scheduler.cpu_off
```

Values

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

Context

The process leaving the cpu.

probe::scheduler.cpu_on

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

Synopsis

```
scheduler.cpu_on
```

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 that was previously running on this cpu

Context

The resuming process.

probe::scheduler.ctxswitch

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

Synopsis

`scheduler.ctxswitch`

Values

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

probe::scheduler.kthread_stop

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

Synopsis

```
scheduler.kthread_stop
```

Values

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

probe::scheduler.kthread_stop.return

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

Synopsis

```
scheduler.kthread_stop.return
```

Values

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

probe::scheduler.migrate

probe::scheduler.migrate — Task migrating across cpus

Synopsis

```
scheduler.migrate
```

Values

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

probe::scheduler.process_exit

probe::scheduler.process_exit — Process exiting

Synopsis

```
scheduler.process_exit
```

Values

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

probe::scheduler.process_fork

probe::scheduler.process_fork — Process forked

Synopsis

`scheduler.process_fork`

Values

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

probe::scheduler.process_free

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

Synopsis

```
scheduler.process_free
```

Values

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

probe::scheduler.process_wait

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

Synopsis

```
scheduler.process_wait
```

Values

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

probe::scheduler.signal_send

probe::scheduler.signal_send — Sending a signal

Synopsis

```
scheduler.signal_send
```

Values

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

probe::scheduler.tick

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

Synopsis

```
scheduler.tick
```

Values

idle boolean indicating whether current is the idle process

name name of the probe point

Context

The process whose accounting will be updated.

probe::scheduler.wait_task

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

Synopsis

```
scheduler.wait_task
```

Values

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

probe::scheduler.wakeup

probe::scheduler.wakeup — Task is woken up

Synopsis

`scheduler.wakeup`

Values

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

probe::scheduler.wakeup_new

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

Synopsis

```
scheduler.wakeup_new
```

Values

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

probe::ioblock.end

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

Synopsis

ioblock.end

Values

<i>bytes_done</i>	number of bytes transferred
<i>rw</i>	binary trace for read/write request
<i>ino</i>	i-node number of the mapped file
<i>phys_segments</i>	number of segments in this bio after physical address coalescing is performed.
<i>opf</i>	operations and flags
<i>sector</i>	beginning sector for the entire bio
<i>flags</i>	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
<i>name</i>	name of the probe point
<i>idx</i>	offset into the bio vector array
<i>error</i>	0 on success
<i>vcnt</i>	bio vector count which represents number of array element (page, offset, length) which makes up this I/O request
<i>size</i>	total size in bytes
<i>hw_segments</i>	number of segments after physical and DMA remapping hardware coalescing is performed
<i>devname</i>	block device name

Context

The process signals the transfer is done.

probe::ioblock.request

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

Synopsis

ioblock.request

Values

<i>hw_segments</i>	number of segments after physical and DMA remapping hardware coalescing is performed
<i>size</i>	total size in bytes
<i>devname</i>	block device name
<i>vcnt</i>	bio vector count which represents number of array element (page, offset, length) which make up this I/O request
<i>p_start_sect</i>	points to the start sector of the partition structure of the device
<i>name</i>	name of the probe point
<i>flags</i>	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
<i>idx</i>	offset into the bio vector array
<i>opf</i>	operations and flags
<i>phys_segments</i>	number of segments in this bio after physical address coalescing is performed
<i>sector</i>	beginning sector for the entire bio
<i>ino</i>	i-node number of the mapped file
<i>rw</i>	binary trace for read/write request
<i>bdev</i>	target block device
<i>bdev_contains</i>	points to the device object which contains the partition (when bio structure represents a partition)

Context

The process makes block I/O request

probe::ioblock_trace.bounce

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

<i>p_start_sect</i>	points to the start sector of the partition structure of the device
<i>q</i>	request queue on which this bio was queued.
<i>vcnt</i>	bio vector count which represents number of array element (page, offset, length) which makes up this I/O request
<i>size</i>	total size in bytes
<i>devname</i>	device for which a buffer bounce was needed.
<i>bdev_contains</i>	points to the device object which contains the partition (when bio structure represents a partition)
<i>bdev</i>	target block device
<i>bytes_done</i>	number of bytes transferred
<i>ino</i>	i-node number of the mapped file
<i>rw</i>	binary trace for read/write request
<i>sector</i>	beginning sector for the entire bio
<i>opf</i>	operations and flags
<i>flags</i>	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
<i>name</i>	name of the probe point
<i>idx</i>	offset into the bio vector array <i>phys_segments</i> - number of segments in this bio after physical address coalescing is performed.

Context

The process creating a block IO request.

probe::ioblock_trace.end

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

Synopsis

```
ioblock_trace.end
```

Values

<i>p_start_sect</i>	points to the start sector of the partition structure of the device
<i>q</i>	request queue on which this bio was queued.
<i>vcnt</i>	bio vector count which represents number of array element (page, offset, length) which makes up this I/O request
<i>size</i>	total size in bytes
<i>devname</i>	block device name
<i>bdev_contains</i>	points to the device object which contains the partition (when bio structure represents a partition)
<i>bdev</i>	target block device
<i>rw</i>	binary trace for read/write request
<i>ino</i>	i-node number of the mapped file
<i>bytes_done</i>	number of bytes transferred
<i>sector</i>	beginning sector for the entire bio
<i>opf</i>	operations and flags
<i>idx</i>	offset into the bio vector array <i>phys_segments</i> - number of segments in this bio after physical address coalescing is performed.
<i>flags</i>	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
<i>name</i>	name of the probe point

Context

The process signals the transfer is done.

probe::ioblock_trace.request

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

Synopsis

ioblock_trace.request

Values

<i>q</i>	request queue on which this bio was queued.
<i>p_start_sect</i>	points to the start sector of the partition structure of the device
<i>size</i>	total size in bytes
<i>devname</i>	block device name
<i>vcnt</i>	bio vector count which represents number of array element (page, offset, length) which make up this I/O request
<i>ino</i>	i-node number of the mapped file
<i>rw</i>	binary trace for read/write request
<i>bytes_done</i>	number of bytes transferred
<i>bdev</i>	target block device
<i>bdev_contains</i>	points to the device object which contains the partition (when bio structure represents a partition)
<i>idx</i>	offset into the bio vector array <i>phys_segments</i> - number of segments in this bio after physical address coalescing is performed.
<i>name</i>	name of the probe point
<i>flags</i>	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
<i>sector</i>	beginning sector for the entire bio
<i>opf</i>	operations and flags

Context

The process makes block I/O request

probe::ioscheduler.elv_add_request

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

Synopsis

```
ioscheduler.elv_add_request
```

Values

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

probe::ioscheduler.elv_add_request.kp

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

Synopsis

```
ioscheduler.elv_add_request.kp
```

Values

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

probe::ioscheduler.elv_add_request.tp

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

probe::ioscheduler.elv_completed_request

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

Synopsis

```
ioscheduler.elv_completed_request
```

Values

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

probe::ioscheduler.elv_next_request

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

Synopsis

```
ioscheduler.elv_next_request
```

Values

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

probe::ioscheduler.elv_next_request.return

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

Synopsis

```
ioscheduler.elv_next_request.return
```

Values

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

probe::ioscheduler_trace.elv_abort_request

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

Synopsis

```
ioscheduler_trace.elv_abort_request
```

Values

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

probe::ioscheduler_trace.elv_completed_request

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

Synopsis

```
ioscheduler_trace.elv_completed_request
```

Values

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

Description

completed.

probe::ioscheduler_trace.elv_issue_request

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

Synopsis

```
ioscheduler_trace.elv_issue_request
```

Values

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

Description

scheduled.

probe::ioscheduler_trace.elv_requeue_request

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

Synopsis

```
ioscheduler_trace.elv_requeue_request
```

Values

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

Description

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

probe::ioscheduler_trace.plugin

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

Synopsis

```
ioscheduler_trace.plugin
```

Values

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

Description

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

probe::ioscheduler_trace.unplug_io

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

Synopsis

```
ioscheduler_trace.unplug_io
```

Values

<i>rq_queue</i>	request queue
<i>name</i>	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.

probe::ioscheduler_trace.unplug_timer

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

Synopsis

```
ioscheduler_trace.unplug_timer
```

Values

<i>name</i>	Name of the probe point
-------------	-------------------------

<i>rq_queue</i>	request queue
-----------------	---------------

Description

with a request queue expires.

Chapter 10. SCSI Tapset

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

probe::scsi.iocompleted

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

Synopsis

`scsi.iocompleted`

Values

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

probe::scsi.iodispatching

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

Synopsis

`scsi.iodispatching`

Values

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

probe::scsi.iodone

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

Synopsis

`scsi.iodone`

Values

<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>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>lun</i>	The lun number
<i>data_direction_str</i>	Data direction, as a string
<i>device_state_str</i>	The current state of the device, as a string
<i>channel</i>	The channel number
<i>host_no</i>	The host number
<i>dev_id</i>	The scsi device id

probe::scsi.ioentry

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

Synopsis

```
scsi.ioentry
```

Values

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

probe::scsi.ioexecute

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

Synopsis

`scsi.ioexecute`

Values

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

probe::scsi.set_state

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

Synopsis

```
scsi.set_state
```

Values

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

Chapter 11. TTY Tapset

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

probe::tty.init

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

Synopsis

```
tty.init
```

Values

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

probe::tty.ioctl

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

Synopsis

```
tty.ioctl
```

Values

<i>name</i>	the file name
<i>arg</i>	the ioctl argument
<i>cmd</i>	the ioctl command

probe::tty.open

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

Synopsis

`tty.open`

Values

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

probe::tty.poll

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

Synopsis

```
tty.poll
```

Values

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

probe::tty.read

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

Synopsis

`tty.read`

Values

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

probe::tty.receive

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

Synopsis

```
tty.receive
```

Values

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

probe::tty.register

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

Synopsis

```
tty.register
```

Values

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

probe::tty.release

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

Synopsis

```
tty.release
```

Values

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

probe::tty.resize

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

Synopsis

```
tty.resize
```

Values

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

probe::tty.unregister

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

Synopsis

```
tty.unregister
```

Values

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

probe::tty.write

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

Synopsis

```
tty.write
```

Values

<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
<i>driver_name</i>	the driver name

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:

probe::irq_handler.entry

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

Synopsis

```
irq_handler.entry
```

Values

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

probe::irq_handler.exit

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

Synopsis

`irq_handler.exit`

Values

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

probe::softirq.entry

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

Synopsis

```
softirq.entry
```

Values

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

probe::softirq.exit

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

Synopsis

```
softirq.exit
```

Values

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

probe::workqueue.create

probe::workqueue.create — Creating a new workqueue

Synopsis

`workqueue.create`

Values

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

probe::workqueue.destroy

probe::workqueue.destroy — Destroying workqueue

Synopsis

```
workqueue.destroy
```

Values

wq_thread task_struct of the workqueue thread

probe::workqueue.execute

probe::workqueue.execute — Executing deferred work

Synopsis

```
workqueue.execute
```

Values

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

probe::workqueue.insert

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

Synopsis

```
workqueue.insert
```

Values

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

Chapter 13. Networking Tapset

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

function::format_ipaddr

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

Synopsis

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

Arguments

<i>addr</i>	the IP address
<i>family</i>	the IP address family (either AF_INET or AF_INET6)

function::htonl

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

Synopsis

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

Arguments

x Value to convert

function::htonll

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

Synopsis

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

Arguments

x Value to convert

function::htons

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

Synopsis

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

Arguments

x Value to convert

function::ip_ntop

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

function::ntohl

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

Synopsis

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

Arguments

x Value to convert

function::ntohl

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

Synopsis

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

Arguments

x Value to convert

function::ntohs

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

Synopsis

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

Arguments

x Value to convert

probe::netdev.change_mac

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

Synopsis

netdev.change_mac

Values

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

probe::netdev.change_mtu

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

Synopsis

`netdev.change_mtu`

Values

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

probe::netdev.change_rx_flag

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

Synopsis

```
netdev.change_rx_flag
```

Values

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

probe::netdev.close

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

Synopsis

```
netdev.close
```

Values

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

probe::netdev.get_stats

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

Synopsis

```
netdev.get_stats
```

Values

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

probe::netdev.hard_transmit

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

Synopsis

netdev.hard_transmit

Values

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

probe::netdev.ioctl

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)

probe::netdev.open

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

Synopsis

`netdev.open`

Values

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

probe::netdev.receive

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

Synopsis

`netdev.receive`

Values

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

probe::netdev.register

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

Synopsis

```
netdev.register
```

Values

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

probe::netdev.rx

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

Synopsis

```
netdev.rx
```

Values

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

probe::netdev.set_promiscuity

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

Synopsis

```
netdev.set_promiscuity
```

Values

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

probe::netdev.transmit

probe::netdev.transmit — Network device transmitting buffer

Synopsis

```
netdev.transmit
```

Values

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

probe::netdev.unregister

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

Synopsis

```
netdev.unregister
```

Values

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

probe::netfilter.arp.forward

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

Synopsis

netfilter.arp.forward

Values

<i>ar_hrd</i>	Format of hardware address
<i>ar_data</i>	Address of ARP packet data region (after the header)
<i>ar_sip</i>	Ethernet+IP only (ar_pro==0x800): source IP address
<i>data_hex</i>	A hexadecimal string representing the packet buffer contents
<i>arphdr</i>	Address of ARP header
<i>data_str</i>	A string representing the packet buffer contents
<i>ar_pro</i>	Format of protocol address
<i>indev</i>	Address of net_device representing input device, 0 if unknown
<i>ar_hln</i>	Length of hardware address
<i>nf_stop</i>	Constant used to signify a 'stop' verdict
<i>ar_tha</i>	Ethernet+IP only (ar_pro==0x800): target hardware (MAC) address
<i>ar_sha</i>	Ethernet+IP only (ar_pro==0x800): source hardware (MAC) address
<i>ar_pln</i>	Length of protocol address
<i>nf_stolen</i>	Constant used to signify a 'stolen' 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>nf_drop</i>	Constant used to signify a 'drop' verdict
<i>nf_accept</i>	Constant used to signify an 'accept' verdict
<i>outdev_name</i>	Name of network device packet will be routed to (if known)
<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>ar_op</i>	ARP opcode (command)
<i>length</i>	The length of the packet buffer contents, in bytes
<i>outdev</i>	Address of net_device representing output device, 0 if unknown

pf

Protocol family -- always “arp”

probe::netfilter.arp.in

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

Synopsis

netfilter.arp.in

Values

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

length

The length of the packet buffer contents, in bytes

probe::netfilter.arp.out

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

Synopsis

netfilter.arp.out

Values

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

length

The length of the packet buffer contents, in bytes

probe::netfilter.bridge.forward

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

Synopsis

netfilter.bridge.forward

Values

<i>outdev</i>	Address of net_device representing output device, 0 if unknown
<i>br_msg</i>	Message age in 1/256 secs
<i>br_hptime</i>	Hello time in 1/256 secs
<i>br_fd</i>	Forward delay in 1/256 secs
<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>length</i>	The length of the packet buffer contents, in bytes
<i>br_bid</i>	Identity of bridge
<i>br_rmac</i>	Root bridge MAC address
<i>outdev_name</i>	Name of network device packet will be routed to (if known)
<i>nf_accept</i>	Constant used to signify an 'accept' verdict
<i>llcproto_stp</i>	Constant used to signify Bridge Spanning Tree Protocol packet
<i>nf_stolen</i>	Constant used to signify a 'stolen' verdict
<i>br_flags</i>	BPDU flags
<i>nf_stop</i>	Constant used to signify a 'stop' verdict
<i>data_str</i>	A string representing the packet buffer contents
<i>data_hex</i>	A hexadecimal string representing the packet buffer contents
<i>br_rid</i>	Identity of root bridge
<i>br_type</i>	BPDU type
<i>br_mac</i>	Bridge MAC address
<i>pf</i>	Protocol family -- always "bridge"
<i>protocol</i>	Packet protocol
<i>nf_repeat</i>	Constant used to signify a 'repeat' verdict
<i>nf_drop</i>	Constant used to signify a 'drop' verdict

<i>br_max</i>	Max age in 1/256 secs
<i>br_poid</i>	Port identifier
<i>brhdr</i>	Address of bridge header
<i>br_cost</i>	Total cost from transmitting bridge to root
<i>br_vid</i>	Protocol version identifier
<i>br_prid</i>	Protocol identifier
<i>indev</i>	Address of net_device representing input device, 0 if unknown
<i>llcpdu</i>	Address of LLC Protocol Data Unit

probe::netfilter.bridge.local_in

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

Synopsis

netfilter.bridge.local_in

Values

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

<i>indev</i>	Address of net_device representing input device, 0 if unknown
<i>br_vid</i>	Protocol version identifier
<i>llcpdu</i>	Address of LLC Protocol Data Unit
<i>protocol</i>	Packet protocol
<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_drop</i>	Constant used to signify a 'drop' verdict

probe::netfilter.bridge.local_out

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

Synopsis

netfilter.bridge.local_out

Values

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

<i>indev</i>	Address of net_device representing input device, 0 if unknown
<i>br_prid</i>	Protocol identifier
<i>br_vid</i>	Protocol version identifier
<i>pf</i>	Protocol family -- always "bridge"
<i>br_mac</i>	Bridge MAC address
<i>protocol</i>	Packet protocol
<i>nf_drop</i>	Constant used to signify a 'drop' verdict
<i>nf_repeat</i>	Constant used to signify a 'repeat' verdict

probe::netfilter.bridge.post_routing

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

Synopsis

netfilter.bridge.post_routing

Values

<i>br_type</i>	BPDU type
<i>br_rid</i>	Identity of root bridge
<i>data_hex</i>	A hexadecimal string representing the packet buffer contents
<i>data_str</i>	A string representing the packet buffer contents
<i>nf_stop</i>	Constant used to signify a 'stop' verdict
<i>br_flags</i>	BPDU flags
<i>nf_stolen</i>	Constant used to signify a 'stolen' verdict
<i>llcproto_stp</i>	Constant used to signify Bridge Spanning Tree Protocol packet
<i>outdev_name</i>	Name of network device packet will be routed to (if known)
<i>nf_accept</i>	Constant used to signify an 'accept' verdict
<i>br_rmac</i>	Root bridge MAC address
<i>br_bid</i>	Identity of bridge
<i>length</i>	The length of the packet buffer contents, in bytes
<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>br_fd</i>	Forward delay in 1/256 secs
<i>br_hptime</i>	Hello time in 1/256 secs
<i>br_msg</i>	Message age in 1/256 secs
<i>outdev</i>	Address of net_device representing output device, 0 if unknown
<i>llcpdu</i>	Address of LLC Protocol Data Unit
<i>br_vid</i>	Protocol version identifier
<i>indev</i>	Address of net_device representing input device, 0 if unknown
<i>br_prid</i>	Protocol identifier
<i>brhdr</i>	Address of bridge header

<i>br_cost</i>	Total cost from transmitting bridge to root
<i>br_poid</i>	Port identifier
<i>br_max</i>	Max age in 1/256 secs
<i>nf_drop</i>	Constant used to signify a 'drop' verdict
<i>nf_repeat</i>	Constant used to signify a 'repeat' verdict
<i>protocol</i>	Packet protocol
<i>pf</i>	Protocol family -- always “bridge”
<i>br_mac</i>	Bridge MAC address

probe::netfilter.bridge.pre_routing

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

Synopsis

netfilter.bridge.pre_routing

Values

<i>br_vid</i>	Protocol version identifier
<i>br_prid</i>	Protocol identifier
<i>indev</i>	Address of net_device representing input device, 0 if unknown
<i>llcpdu</i>	Address of LLC Protocol Data Unit
<i>br_cost</i>	Total cost from transmitting bridge to root
<i>brhdr</i>	Address of bridge header
<i>br_max</i>	Max age in 1/256 secs
<i>br_poid</i>	Port identifier
<i>nf_repeat</i>	Constant used to signify a 'repeat' verdict
<i>nf_drop</i>	Constant used to signify a 'drop' verdict
<i>protocol</i>	Packet protocol
<i>br_mac</i>	Bridge MAC address
<i>pf</i>	Protocol family -- always "bridge"
<i>data_hex</i>	A hexadecimal string representing the packet buffer contents
<i>data_str</i>	A string representing the packet buffer contents
<i>br_rid</i>	Identity of root bridge
<i>br_type</i>	BPDU type
<i>br_flags</i>	BPDU flags
<i>nf_stop</i>	Constant used to signify a 'stop' verdict
<i>llcproto_stp</i>	Constant used to signify Bridge Spanning Tree Protocol packet
<i>nf_stolen</i>	Constant used to signify a 'stolen' verdict
<i>br_rmac</i>	Root bridge MAC address
<i>nf_accept</i>	Constant used to signify an 'accept' verdict
<i>outdev_name</i>	Name of network device packet will be routed to (if known)

<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>br_bid</i>	Identity of bridge
<i>length</i>	The length of the packet buffer contents, in bytes
<i>br_fd</i>	Forward delay in 1/256 secs
<i>br_msg</i>	Message age in 1/256 secs
<i>outdev</i>	Address of net_device representing output device, 0 if unknown
<i>br_hptime</i>	Hello time in 1/256 secs

probe::netfilter.ip.forward

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

Synopsis

netfilter.ip.forward

Values

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

<i>family</i>	IP address family
<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>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

probe::netfilter.ip.local_in

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

Synopsis

netfilter.ip.local_in

Values

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

<i>data_hex</i>	A hexadecimal string representing the packet buffer contents
<i>data_str</i>	A string representing the packet buffer contents
<i>saddr</i>	A string representing the source IP address
<i>indev</i>	Address of net_device representing input device, 0 if unknown
<i>psh</i>	TCP PSH flag (if protocol is TCP; ipv4 only)

probe::netfilter.ip.local_out

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

Synopsis

netfilter.ip.local_out

Values

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

<i>indev</i>	Address of net_device representing input device, 0 if unknown
<i>saddr</i>	A string representing the source IP address
<i>data_str</i>	A string representing the packet buffer contents
<i>rst</i>	TCP RST flag (if protocol is TCP; ipv4 only)
<i>data_hex</i>	A hexadecimal string representing the packet buffer contents

probe::netfilter.ip.post_routing

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

Synopsis

netfilter.ip.post_routing

Values

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

<i>length</i>	The length of the packet buffer contents, in bytes
<i>ipproto_udp</i>	Constant used to signify that the packet protocol is UDP
<i>family</i>	IP address family
<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)

probe::netfilter.ip.pre_routing

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

Synopsis

netfilter.ip.pre_routing

Values

<i>nf_stolen</i>	Constant used to signify a 'stolen' verdict
<i>iphdr</i>	Address of IP header
<i>nf_stop</i>	Constant used to signify a 'stop' verdict
<i>daddr</i>	A string representing the destination IP address
<i>fin</i>	TCP FIN flag (if protocol is TCP; ipv4 only)
<i>psb</i>	TCP PSB flag (if protocol is TCP; ipv4 only)
<i>indev</i>	Address of net_device representing input device, 0 if unknown
<i>saddr</i>	A string representing the source IP address
<i>data_str</i>	A string representing the packet buffer contents
<i>rst</i>	TCP RST flag (if protocol is TCP; ipv4 only)
<i>data_hex</i>	A hexadecimal string representing the packet buffer contents
<i>ack</i>	TCP ACK flag (if protocol is TCP; ipv4 only)
<i>pf</i>	Protocol family - either 'ipv4' or 'ipv6'
<i>sport</i>	TCP or UDP source port (ipv4 only)
<i>syn</i>	TCP SYN flag (if protocol is TCP; ipv4 only)
<i>outdev</i>	Address of net_device representing output device, 0 if unknown
<i>protocol</i>	Packet protocol from driver (ipv4 only)
<i>urg</i>	TCP URG flag (if protocol is TCP; ipv4 only)
<i>dport</i>	TCP or UDP destination port (ipv4 only)
<i>length</i>	The length of the packet buffer contents, in bytes
<i>family</i>	IP address family
<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>ipproto_udp</i>	Constant used to signify that the packet protocol is UDP

<i>nf_accept</i>	Constant used to signify an 'accept' verdict
<i>outdev_name</i>	Name of network device packet will be routed to (if known)
<i>nf_drop</i>	Constant used to signify a 'drop' verdict
<i>ipproto_tcp</i>	Constant used to signify that the packet protocol is TCP
<i>nf_repeat</i>	Constant used to signify a 'repeat' verdict

probe::sunrpc.clnt.bind_new_program

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

Synopsis

```
sunrpc.clnt.bind_new_program
```

Values

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

probe::sunrpc.clnt.call_async

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

Synopsis

`sunrpc.clnt.call_async`

Values

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

probe::sunrpc.clnt.call_sync

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

Synopsis

`sunrpc.clnt.call_sync`

Values

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

probe::sunrpc.clnt.clone_client

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

Synopsis

```
sunrpc.clnt.clone_client
```

Values

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

probe::sunrpc.clnt.create_client

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

Synopsis

`sunrpc.clnt.create_client`

Values

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

probe::sunrpc.clnt.restart_call

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

Synopsis

```
sunrpc.clnt.restart_call
```

Values

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

probe::sunrpc.clnt.shutdown_client

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

Synopsis

`sunrpc.clnt.shutdown_client`

Values

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

probe::sunrpc.sched.delay

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

Synopsis

`sunrpc.sched.delay`

Values

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

probe::sunrpc.sched.execute

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

Synopsis

`sunrpc.sched.execute`

Values

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

probe::sunrpc.sched.new_task

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

Synopsis

`sunrpc.sched.new_task`

Values

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

probe::sunrpc.sched.release_task

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

Synopsis

`sunrpc.sched.release_task`

Values

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

probe::sunrpc.svc.create

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

Synopsis

```
sunrpc.svc.create
```

Values

<i>prog</i>	the number of the program
<i>progname</i>	the name of the program
<i>pg_nvers</i>	the number of supported versions
<i>bufsize</i>	the buffer size

probe::sunrpc.svc.destroy

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

Synopsis

`sunrpc.svc.destroy`

Values

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

probe::sunrpc.svc.drop

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

Synopsis

`sunrpc.svc.drop`

Values

<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
<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_name</i>	the service name

probe::sunrpc.svc.process

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

Synopsis

`sunrpc.svc.process`

Values

<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
<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_name</i>	the service name
<i>sv_prog</i>	the number of the program
<i>sv_nthreads</i>	the number of concurrent threads

probe::sunrpc.svc.recv

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_nrthreads</i>	the number of concurrent threads
<i>sv_prog</i>	the number of the program
<i>timeout</i>	the timeout of waiting for data

probe::sunrpc.svc.register

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

Synopsis

```
sunrpc.svc.register
```

Values

<i>sv_name</i>	the service name
<i>port</i>	the port number
<i>prog</i>	the number of the program
<i>progname</i>	the name of the program
<i>prot</i>	the IP protocol number

Description

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

probe::sunrpc.svc.send

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

Synopsis

`sunrpc.svc.send`

Values

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

probe::tcp.disconnect

probe::tcp.disconnect — TCP socket disconnection

Synopsis

`tcp.disconnect`

Values

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

Context

The process which disconnects tcp

probe::tcp.disconnect.return

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

Synopsis

```
tcp.disconnect.return
```

Values

<i>name</i>	Name of this probe
<i>ret</i>	Error code (0: no error)

Context

The process which disconnects tcp

probe::tcp.receive

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

Synopsis

tcp.receive

Values

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

probe::tcp.recvmsg

probe::tcp.recvmsg — Receiving TCP message

Synopsis

`tcp.recvmsg`

Values

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

Context

The process which receives a tcp message

probe::tcp.recvmsg.return

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

Synopsis

```
tcp.recvmsg.return
```

Values

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

Context

The process which receives a tcp message

probe::tcp.sendmsg

probe::tcp.sendmsg — Sending a tcp message

Synopsis

`tcp.sendmsg`

Values

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

Context

The process which sends a tcp message

probe::tcp.sendmsg.return

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

Synopsis

```
tcp.sendmsg.return
```

Values

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

name Name of this probe

Context

The process which sends a tcp message

probe::tcp.setsockopt

probe::tcp.setsockopt — Call to `setsockopt`

Synopsis

`tcp.setsockopt`

Values

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

Context

The process which calls `setsockopt`

probe::tcp.setsockopt.return

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

probe::udp.disconnect

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

Synopsis

udp.disconnect

Values

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

Context

The process which requests a UDP disconnection

probe::udp.disconnect.return

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

Synopsis

```
udp.disconnect.return
```

Values

<i>daddr</i>	A string representing the destination IP address
<i>ret</i>	Error code (0: no error)
<i>saddr</i>	A string representing the source IP address
<i>family</i>	IP address family
<i>sport</i>	UDP source port
<i>dport</i>	UDP destination port
<i>name</i>	The name of this probe

Context

The process which requested a UDP disconnection

probe::udp.recvmsg

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

Synopsis

`udp.recvmsg`

Values

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

Context

The process which received a UDP message

probe::udp.recvmsg.return

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

Synopsis

`udp.recvmsg.return`

Values

<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>family</i>	IP address family
<i>size</i>	Number of bytes received by the process
<i>saddr</i>	A string representing the source IP address

Context

The process which received a UDP message

probe::udp.sendmsg

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

Context

The process which sent a UDP message

probe::udp.sendmsg.return

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

Synopsis

```
udp.sendmsg.return
```

Values

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

Context

The process which sent a UDP message

Chapter 14. Socket Tapset

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

function::inet_get_ip_source

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

function::inet_get_local_port

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

function::sock_fam_num2str

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

function::sock_fam_str2num

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

function::sock_prot_num2str

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

Synopsis

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

Arguments

proto The protocol number

function::sock_prot_str2num

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

function::sock_state_num2str

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

function::sock_state_str2num

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

probe::socket.aio_read

probe::socket.aio_read — Receiving message via sock_aio_read

Synopsis

```
socket.aio_read
```

Values

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

Context

The message sender

Description

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

probe::socket.aio_read.return

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>flags</i>	Socket flags value
<i>family</i>	Protocol family value
<i>type</i>	Socket type value
<i>state</i>	Socket state 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_aio_read` function

probe::socket.aio_write

probe::socket.aio_write — Message send via sock_aio_write

Synopsis

```
socket.aio_write
```

Values

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

Context

The message sender

Description

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

probe::socket.aio_write.return

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

Synopsis

```
socket.aio_write.return
```

Values

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

Context

The message receiver.

Description

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

probe::socket.close

probe::socket.close — Close a socket

Synopsis

```
socket.close
```

Values

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

Context

The requester (user process or kernel)

Description

Fires at the beginning of closing a socket.

probe::socket.close.return

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.

probe::socket.create

probe::socket.create — Creation of a socket

Synopsis

```
socket.create
```

Values

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

Context

The requester (see requester variable)

Description

Fires at the beginning of creating a socket.

probe::socket.create.return

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

Synopsis

`socket.create.return`

Values

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

Context

The requester (user process or kernel)

Description

Fires at the conclusion of creating a socket.

probe::socket.read_iter

probe::socket.read_iter — Receiving message via `sock_read_iter`

Synopsis

```
socket.read_iter
```

Values

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

Context

The message sender

Description

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

probe::socket.read_iter.return

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

Synopsis

```
socket.read_iter.return
```

Values

<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>state</i>	Socket state value
<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

Context

The message receiver.

Description

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

probe::socket.readv

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

Synopsis

`socket.readv`

Values

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

Context

The message sender

Description

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

probe::socket.readv.return

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

Synopsis

`socket.readv.return`

Values

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

Context

The message receiver.

Description

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

probe::socket.receive

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

Synopsis

```
socket.receive
```

Values

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

Context

The message receiver

probe::socket.recvmsg

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

Synopsis

`socket.recvmsg`

Values

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

Context

The message receiver.

Description

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

probe::socket.recvmsg.return

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

Synopsis

`socket.recvmsg.return`

Values

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

Context

The message receiver.

Description

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

probe::socket.send

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

Synopsis

```
socket.send
```

Values

<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>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>state</i>	Socket state value
<i>protocol</i>	Protocol value

Context

The message sender

probe::socket.sendmsg

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

Synopsis

```
socket.sendmsg
```

Values

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

Context

The message sender

Description

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

probe::socket.sendmsg.return

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

Synopsis

`socket.sendmsg.return`

Values

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

probe::socket.write_iter

probe::socket.write_iter — Message send via `sock_write_iter`

Synopsis

```
socket.write_iter
```

Values

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

Context

The message sender

Description

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

probe::socket.write_iter.return

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

Synopsis

```
socket.write_iter.return
```

Values

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

Context

The message receiver.

Description

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

probe::socket.writev

probe::socket.writev — Message sent via socket_writev

Synopsis

```
socket.writev
```

Values

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

Context

The message sender

Description

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

probe::socket.writev.return

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

Synopsis

```
socket.writev.return
```

Values

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

function::ipmib_filter_key

function::ipmib_filter_key — Default filter function for ipmib.* probes

Synopsis

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

Arguments

<i>skb</i>	pointer to the struct <i>sk_buff</i>
<i>op</i>	value to be counted if <i>skb</i> passes the filter
<i>SourceIsLocal</i>	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.

function::ipmib_get_proto

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*.

function::ipmib_local_addr

function::ipmib_local_addr — Get the local ip address

Synopsis

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

Arguments

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

Description

Returns the local ip address *skb*.

function::ipmib_remote_addr

function::ipmib_remote_addr — Get the remote ip address

Synopsis

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

Arguments

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

Description

Returns the remote ip address from *skb*.

function::ipmib_tcp_local_port

function::ipmib_tcp_local_port — Get the local tcp port

Synopsis

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

Arguments

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

Description

Returns the local tcp port from *skb*.

function::ipmib_tcp_remote_port

function::ipmib_tcp_remote_port — Get the remote tcp port

Synopsis

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

Arguments

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

Description

Returns the remote tcp port from *skb*.

function::linuxmib_filter_key

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.

function::tcpmib_filter_key

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.

function::tcpmib_get_state

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.

function::tcpmib_local_addr

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.

function::tcpmib_local_port

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.

function::tcpmib_remote_addr

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.

function::tcpmib_remote_port

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.

probe::ipmib.ForwDatagrams

probe::ipmib.ForwDatagrams — Count forwarded packet

Synopsis

```
ipmib.ForwDatagrams
```

Values

skb pointer to the struct `sk_buff` being acted on

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

Description

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

probe::ipmib.FragFails

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)

probe::ipmib.FragOKs

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)

probe::ipmib.InAddrErrors

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 is counted in the global *InAddrErrors* (equivalent to SNMP's MIB `IPSTATS_MIB_INADDRERRORS`)

probe::ipmib.InDiscards

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`)

probe::ipmib.InNoRoutes

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)

probe::ipmib.InReceives

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`)

probe::ipmib.InUnknownProtos

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

Synopsis

```
ipmib.InUnknownProtos
```

Values

skb pointer to the struct `sk_buff` being acted on

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

Description

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

probe::ipmib.OutRequests

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`)

probe::ipmib.ReasmReqds

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

Synopsis

```
ipmib.ReasmReqds
```

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 *ReasmReqds* (equivalent to SNMP's MIB IPSTATS_MIB_REASMREQDS)

probe::ipmib.ReasmTimeout

probe::ipmib.ReasmTimeout — Count Reassembly Timeouts

Synopsis

```
ipmib.ReasmTimeout
```

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

probe::linuxmib.DelayedACKs

probe::linuxmib.DelayedACKs — Count of delayed acks

Synopsis

`linuxmib.DelayedACKs`

Values

sk Pointer to the struct sock being acted on

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

Description

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

probe::linuxmib.ListenDrops

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

Synopsis

```
linuxmib.ListenDrops
```

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 `linuxmib_filter_key`. If the packet passes the filter is counted in the global *ListenDrops* (equivalent to SNMP's MIB `LINUX_MIB_LISTENDROPS`)

probe::linuxmib.ListenOverflows

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 is counted in the global *ListenOverflows* (equivalent to SNMP's MIB LINUX_MIB_LISTENOVERFLOWS)

probe::linuxmib.TCPMemoryPressures

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

Synopsis

`linuxmib.TCPMemoryPressures`

Values

skb Pointer to the struct sock being acted on

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

Description

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

probe::tcpmib.ActiveOpens

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

Synopsis

```
tcpmib.ActiveOpens
```

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 counted in the global *ActiveOpens* (equivalent to SNMP's MIB TCP_MIB_ACTIVEOPENS)

probe::tcpmib.AttemptFails

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

Synopsis

```
tcpmib.AttemptFails
```

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 *AttemptFails* (equivalent to SNMP's MIB TCP_MIB_ATTEMPTFAILS)

probe::tcpmib.CurrEstab

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 counted in the global *CurrEstab* (equivalent to SNMP's MIB TCP_MIB_CURRESTAB)

probe::tcpmib.EstabResets

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

Synopsis

`tcpmib.EstabResets`

Values

sk pointer to the struct sock being acted on

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

Description

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

probe::tcpmib.InSegs

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

Synopsis

`tcpmib.InSegs`

Values

sk pointer to the struct sock being acted on

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

Description

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

probe::tcpmib.OutRsts

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

Synopsis

```
tcpmib.OutRsts
```

Values

skb pointer to the struct sock being acted on

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

Description

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

probe::tcpmib.OutSegs

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

Synopsis

`tcpmib.OutSegs`

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 *OutSegs* (equivalent to SNMP's MIB TCP_MIB_OUTSEGS)

probe::tcpmib.PassiveOpens

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

Synopsis

```
tcpmib.PassiveOpens
```

Values

sk pointer to the struct sock being acted on

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

Description

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

probe::tcpmib.RetransSegs

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

Synopsis

```
tcpmib.RetransSegs
```

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 *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:

function::get_loadavg_index

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.

function::sprint_loadavg

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.

function::target_set_pid

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.

function::target_set_report

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.

probe::kprocess.create

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

Synopsis

`kprocess.create`

Values

<i>new_tid</i>	The TID of the newly created task
<i>new_pid</i>	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.

probe::kprocess.exec

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

Synopsis

`kprocess.exec`

Values

<i>filename</i>	The path to the new executable
<i>argstr</i>	A string containing the filename followed by the arguments to pass, excluding 0th arg (SystemTap v2.5+)
<i>name</i>	Name of the system call (“execve”) (SystemTap v2.5+)
<i>args</i>	The arguments to pass to the new executable, including the 0th arg (SystemTap v2.5+)

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+.

probe::kprocess.exec_complete

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

Synopsis

```
kprocess.exec_complete
```

Values

<i>success</i>	A boolean indicating whether the exec was successful
<i>errno</i>	The error number resulting from the exec
<i>retstr</i>	A string representation of errno (SystemTap v2.5+)
<i>name</i>	Name of the system call (“execve”) (SystemTap v2.5+)

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+.

probe::kprocess.exit

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.

probe::kprocess.release

probe::kprocess.release — Process released

Synopsis

kprocess.release

Values

<i>task</i>	A task handle to the process being released
<i>released_pid</i>	PID of the process being released
<i>pid</i>	Same as <i>released_pid</i> for compatibility (deprecated)
<i>released_tid</i>	TID of the task 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.

probe::kprocess.start

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:

function::get_sa_flags

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.

function::get_sa_handler

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.

function::is_sig_blocked

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.

function::sa_flags_str

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.

function::sa_handler_str

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.

function::signal_str

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

Synopsis

```
signal_str(num:)
```

Arguments

num the signal number to convert to string.

function::sigset_mask_str

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.

probe::signal.check_ignored

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

Synopsis

```
signal.check_ignored
```

Values

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

probe::signal.check_ignored.return

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

Synopsis

```
signal.check_ignored.return
```

Values

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

probe::signal.checkperm

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

Synopsis

`signal.checkperm`

Values

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

probe::signal.checkperm.return

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

Synopsis

```
signal.checkperm.return
```

Values

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

probe::signal.do_action

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

Synopsis

```
signal.do_action
```

Values

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

probe::signal.do_action.return

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

Synopsis

```
signal.do_action.return
```

Values

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

probe::signal.flush

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

Synopsis

```
signal.flush
```

Values

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

probe::signal.force_segv

probe::signal.force_segv — Forcing send of SIGSEGV

Synopsis

```
signal.force_segv
```

Values

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

probe::signal.force_segv.return

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

Synopsis

```
signal.force_segv.return
```

Values

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

probe::signal.handle

probe::signal.handle — Signal handler being invoked

Synopsis

`signal.handle`

Values

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

probe::signal.handle.return

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

Synopsis

```
signal.handle.return
```

Values

retstr Return value as a string

name Name of the probe point

Description

(deprecated in SystemTap 2.1)

probe::signal.pending

probe::signal.pending — Examining pending signal

Synopsis

```
signal.pending
```

Values

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

Description

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

probe::signal.pending.return

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

Synopsis

```
signal.pending.return
```

Values

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

probe::signal.procmask

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

Synopsis

`signal.procmask`

Values

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

probe::signal.procmask.return

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

Synopsis

```
signal.procmask.return
```

Values

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

probe::signal.send

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

Synopsis

```
signal.send
```

Values

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

Context

The signal's sender.

probe::signal.send.return

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

Synopsis

```
signal.send.return
```

Values

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

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.

probe::signal.send_sig_queue

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

Synopsis

```
signal.send_sig_queue
```

Values

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

probe::signal.send_sig_queue.return

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

Synopsis

```
signal.send_sig_queue.return
```

Values

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

probe::signal.sys_tgkill

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

Synopsis

```
signal.sys_tgkill
```

Values

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

Description

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

probe::signal.sys_tgkill.return

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

Synopsis

```
signal.sys_tgkill.return
```

Values

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

probe::signal.sys_tkill

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

Synopsis

```
signal.sys_tkill
```

Values

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

Description

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

probe::signal.syskill

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

Synopsis

```
signal.syskill
```

Values

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

probe::signal.syskill.return

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

Synopsis

```
signal.syskill.return
```

Values

None

probe::signal.systkill.return

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

Synopsis

```
signal.systkill.return
```

Values

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

probe::signal.wakeup

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

Synopsis

`signal.wakeup`

Values

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

function::errno_str

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.

function::return_str

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

Synopsis

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

Arguments

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

function::returnstr

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`.

function::returnval

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:

function::rlimit_from_str

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:

function::MAJOR

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.

function::MINOR

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.

function::MKDEV

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.

function::usrdev2kerndev

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.

function::d_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).

function::d_path

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.

function::fullpath_struct_file

function::fullpath_struct_file — get the full path

Synopsis

```
fullpath_struct_file:string(task:long,file:long)
```

Arguments

task task_struct pointer.

file Pointer to “struct file”.

Description

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

function::fullpath_struct_nameidata

function::fullpath_struct_nameidata — get the full nameidata path

Synopsis

```
fullpath_struct_nameidata(nd:)
```

Arguments

nd Pointer to “struct nameidata”.

Description

Returns the full dirent name (full path to the root), like the kernel (and systemtap-tapset) `d_path` function, with a “/”.

function::fullpath_struct_path

function::fullpath_struct_path — get the full path

Synopsis

```
fullpath_struct_path:string(path:long)
```

Arguments

path Pointer to “struct path”.

Description

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

function::inode_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.

function::inode_path

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.

function::real_mount

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.

function::reverse_path_walk

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).

function::task_dentry_path

function::task_dentry_path — get the full dentry path

Synopsis

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

Arguments

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

Description

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

Chapter 22. Logging Tapset

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

function::assert

function::assert — evaluate assertion

Synopsis

- 1) `assert(expression:long)`
- 2) `assert(expression:long,msg:string)`

Arguments

<i>expression</i>	The expression to evaluate
<i>msg</i>	The formatted message string

Description

- 1) This function checks the expression and aborts the current running probe if expression evaluates to zero. `Useserror` and may be caught by `try{ } catch{ }`. A default message will be displayed.
- 2) This function checks the expression and aborts the current running probe if expression evaluates to zero. `Useserror` and may be caught by `try{ } catch{ }`. The specified message will be displayed.

function::error

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`.

function::exit

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.

function::ftrace

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.

function::log

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.

function::printk

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.

function::warn

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

Synopsis

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

Arguments

msg The formatted message string

Description

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

Chapter 23. Queue Statistics Tapset

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

function::qs_done

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.

function::qs_run

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.

function::qs_wait

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.

function::qsq_blocked

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.

function::qsq_print

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.

function::qsq_service_time

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.

function::qsq_start

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.

function::qsq_throughput

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

Synopsis

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

Arguments

qname queue name

scale scale variable to take account for interval fraction

Description

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

function::qsq_utilization

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.

function::qsq_wait_queue_length

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

function::qsq_wait_time

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.

function::randint

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.

function::atomic_long_read

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.

function::atomic_read

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.

function::kernel_buffer_quoted

function::kernel_buffer_quoted — Retrieves and quotes buffer from kernel space

Synopsis

- 1) `kernel_buffer_quoted:string(addr:long,inlen:long)`
- 2) `kernel_buffer_quoted:string(addr:long,inlen:long,outlen:long)`

Arguments

- | | |
|---------------|--|
| <i>addr</i> | the kernel space address to retrieve the buffer from |
| <i>inlen</i> | the exact length of the buffer to read |
| <i>outlen</i> | the maximum length of the output string |

Description

1) Reads *inlen* characters of a buffer from the given kernel space memory address, and returns up to MAXSTRINGLEN 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 kernel space data is not accessible at the given address, the address itself is returned as a string, without double quotes.

2) Reads *inlen* characters of a buffer from the given kernel 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 kernel space data is not accessible at the given address, the address itself is returned as a string, without double quotes.

function::kernel_buffer_quoted_error

function::kernel_buffer_quoted_error — Retrieves and quotes buffer from kernel space

Synopsis

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

Arguments

<i>addr</i>	the kernel space address to retrieve the buffer from
<i>inlen</i>	the exact length of the buffer to read
<i>outlen</i>	the maximum length of the output string

Description

Reads *inlen* characters of a buffer from the given kernel 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 kernel space data is not accessible at the given address, an error is thrown.

function::kernel_char

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.

function::kernel_int

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.

function::kernel_long

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.

function::kernel_pointer

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.

function::kernel_short

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.

function::kernel_string

function::kernel_string — Retrieves string from kernel memory

Synopsis

- 1) `kernel_string:string(addr:long)`
- 2) `kernel_string:string(addr:long,err_msg:string)`

Arguments

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

Description

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

function::kernel_string_n

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.

function::kernel_string_quoted

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.

function::kernel_string_quoted_utf16

function::kernel_string_quoted_utf16 — Quote given kernel UTF-16 string.

Synopsis

```
kernel_string_quoted_utf16:string(addr:long)
```

Arguments

addr The kernel address to retrieve the string from

Description

This function combines quoting as per *string_quoted* and UTF-16 decoding as per *kernel_string_utf16*.

function::kernel_string_quoted_utf32

function::kernel_string_quoted_utf32 — Quote given UTF-32 kernel string.

Synopsis

```
kernel_string_quoted_utf32:string(addr:long)
```

Arguments

addr The kernel address to retrieve the string from

Description

This function combines quoting as per *string_quoted* and UTF-32 decoding as per *kernel_string_utf32*.

function::kernel_string_utf16

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

Synopsis

- 1) `kernel_string_utf16:string(addr:long)`
- 2) `kernel_string_utf16:string(addr:long,err_msg:string)`

Arguments

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

Description

- 1) 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.
- 2) 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.

function::kernel_string_utf32

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

Synopsis

- 1) `kernel_string_utf32:string(addr:long)`
- 2) `kernel_string_utf32:string(addr:long,err_msg:string)`

Arguments

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

Description

- 1) 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.
- 2) 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.

function::user_buffer_quoted

function::user_buffer_quoted — Retrieves and quotes buffer from user space

Synopsis

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

Arguments

<i>addr</i>	the user space address to retrieve the buffer from
<i>inlen</i>	the exact length of the buffer to read
<i>outlen</i>	the maximum length of the output string

Description

Reads *inlen* characters of a buffer 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 user space data is not accessible at the given address, the address itself is returned as a string, without double quotes.

function::user_buffer_quoted_error

function::user_buffer_quoted_error — Retrieves and quotes buffer from user space

Synopsis

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

Arguments

<i>addr</i>	the user space address to retrieve the buffer from
<i>inlen</i>	the exact length of the buffer to read
<i>outlen</i>	the maximum length of the output string

Description

Reads *inlen* characters of a buffer 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 user space data is not accessible at the given address, an error is thrown.

function::user_char

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.

function::user_char_error

function::user_char_error — Retrieves a char value stored in user space

Synopsis

```
user_char_error: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. If the user space data is not accessible, an error will occur.

function::user_char_warn

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 data is not accessible and warns about the failure (but does not error).

function::user_int

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.

function::user_int16

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.

function::user_int16_error

function::user_int16_error — Retrieves a 16-bit integer value stored in user space

Synopsis

```
user_int16_error: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. If the user space data is not accessible, an error will occur.

function::user_int32

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.

function::user_int32_error

function::user_int32_error — Retrieves a 32-bit integer value stored in user space

Synopsis

```
user_int32_error: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. If the user space data is not accessible, an error will occur.

function::user_int64

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.

function::user_int64_error

function::user_int64_error — Retrieves a 64-bit integer value stored in user space

Synopsis

```
user_int64_error: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. If the user space data is not accessible, an error will occur.

function::user_int8

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.

function::user_int8_error

function::user_int8_error — Retrieves a 8-bit integer value stored in user space

Synopsis

```
user_int8_error: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. If the user space data is not accessible, an error will occur.

function::user_int_error

function::user_int_error — Retrieves an int value stored in user space

Synopsis

```
user_int_error: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. If the user space data is not accessible, an error will occur.

function::user_int_warn

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 data is not accessible and warns about the failure (but does not error).

function::user_long

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).

function::user_long_error

function::user_long_error — Retrieves a long value stored in user space

Synopsis

```
user_long_error: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. If the user space data is not accessible, an error will occur. 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).

function::user_long_warn

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 data is not accessible and warns about the failure (but does not error). 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).

function::user_short

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.

function::user_short_error

function::user_short_error — Retrieves a short value stored in user space

Synopsis

```
user_short_error: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. If the user space data is not accessible, an error will occur.

function::user_short_warn

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 data is not accessible and warns about the failure (but does not error).

function::user_string

function::user_string — Retrieves string from user space

Synopsis

- 1) `user_string:string(addr:long)`
- 2) `user_string:string(addr:long,err_msg:string)`

Arguments

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

Description

- 1) 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.
- 2) 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.

function::user_string_n

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

Synopsis

- 1) `user_string_n:string(addr:long,n:long)`
- 2) `user_string_n:string(addr:long,n:long,err_msg:string)`

Arguments

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

Description

- 1) 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.
- 2) 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.

function::user_string_n_quoted

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

Synopsis

- 1) `user_string_n_quoted:string(addr:long,n:long)`
- 2) `user_string_n_quoted:string(addr:long,inlen:long,outlen:long)`

Arguments

- | | |
|---------------|---|
| <i>addr</i> | the user space address to retrieve the string from |
| <i>n</i> | the maximum length of the string (if not null terminated) |
| <i>inlen</i> | the maximum length of the string to read (if not null terminated) |
| <i>outlen</i> | the maximum length of the output string |

Description

1) 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.

2) 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.

function::user_string_n_warn

function::user_string_n_warn — Retrieves string from user space

Synopsis

- 1) `user_string_n_warn:string(addr:long,n:long)`
- 2) `user_string_n_warn:string(addr:long,n:long,warn_msg:string)`

Arguments

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

Description

- 1) 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.
- 2) 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.

function::user_string_quoted

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.

function::user_string_quoted_utf16

function::user_string_quoted_utf16 — Quote given user UTF-16 string.

Synopsis

```
user_string_quoted_utf16:string(addr:long)
```

Arguments

addr The user address to retrieve the string from

Description

This function combines quoting as per *string_quoted* and UTF-16 decoding as per *user_string_utf16*.

function::user_string_quoted_utf32

function::user_string_quoted_utf32 — Quote given user UTF-32 string.

Synopsis

```
user_string_quoted_utf32:string(addr:long)
```

Arguments

addr The user address to retrieve the string from

Description

This function combines quoting as per *string_quoted* and UTF-32 decoding as per *user_string_utf32*.

function::user_string_utf16

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

Synopsis

- 1) `user_string_utf16:string(addr:long)`
- 2) `user_string_utf16:string(addr:long,err_msg:string)`

Arguments

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

Description

- 1) 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.
- 2) 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.

function::user_string_utf32

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

Synopsis

- 1) `user_string_utf32:string(addr:long)`
- 2) `user_string_utf32:string(addr:long,err_msg:string)`

Arguments

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

Description

- 1) 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.
- 2) 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.

function::user_string_warn

function::user_string_warn — Retrieves string from user space

Synopsis

- 1) `user_string_warn:string(addr:long)`
- 2) `user_string_warn:string(addr:long, warn_msg:string)`

Arguments

- | | |
|-----------------|---|
| <i>addr</i> | the user space address to retrieve the string from |
| <i>warn_msg</i> | the warning message to return when data isn't available |

Description

- 1) 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.
- 2) 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.

function::user_uint16

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.

function::user_uint16_error

function::user_uint16_error — Retrieves an unsigned 16-bit integer value stored in user space

Synopsis

```
user_uint16_error: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. If the user space data is not accessible, an error will occur.

function::user_uint32

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.

function::user_uint32_error

function::user_uint32_error — Retrieves an unsigned 32-bit integer value stored in user space

Synopsis

```
user_uint32_error: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. If the user space data is not accessible, an error will occur.

function::user_uint64

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.

function::user_uint64_error

function::user_uint64_error — Retrieves an unsigned 64-bit integer value stored in user space

Synopsis

```
user_uint64_error: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. If the user space data is not accessible, an error will occur.

function::user_uint8

function::user_uint8 — Retrieves a 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.

function::user_uint8_error

function::user_uint8_error — Retrieves a unsigned 8-bit integer value stored in user space

Synopsis

```
user_uint8_error: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. If the user space data is not accessible, an error will occur.

function::user_ulong

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).

function::user_ulong_error

function::user_ulong_error — Retrieves a unsigned long value stored in user space

Synopsis

```
user_ulong_error: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. If the user space data is not accessible, an error will occur. 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).

function::user_ulong_warn

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 data is not accessible and warns about the failure (but does not error). 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).

function::user_ushort

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.

function::user_ushort_error

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

Synopsis

```
user_ushort_error: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. If the user space data is not accessible, an error will occur.

function::user_ushort_warn

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 data is not accessible and warns about the failure (but does not error).

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).

function::set_kernel_char

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).

function::set_kernel_int

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).

function::set_kernel_long

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).

function::set_kernel_pointer

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).

function::set_kernel_short

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).

function::set_kernel_string

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).

function::set_kernel_string_n

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

<i>addr</i>	The kernel address to write the string to
<i>n</i>	The maximum length of the string
<i>val</i>	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).

function::set_user_char

function::set_user_char — Writes a char value to user memory

Synopsis

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

Arguments

addr The user address to write the char to

val The char which is to be written

Description

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

function::set_user_int

function::set_user_int — Writes an int value to user memory

Synopsis

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

Arguments

addr The user address to write the int to

val The int which is to be written

Description

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

function::set_user_long

function::set_user_long — Writes a long value to user memory

Synopsis

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

Arguments

addr The user address to write the long to

val The long which is to be written

Description

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

function::set_user_pointer

function::set_user_pointer — Writes a pointer value to user memory.

Synopsis

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

Arguments

addr The user address to write the pointer to

val The pointer which is to be written

Description

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

function::set_user_short

function::set_user_short — Writes a short value to user memory

Synopsis

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

Arguments

addr The user address to write the short to

val The short which is to be written

Description

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

function::set_user_string

function::set_user_string — Writes a string to user memory

Synopsis

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

Arguments

addr The user address to write the string to

val The string which is to be written

Description

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

function::set_user_string_n

function::set_user_string_n — Writes a string of given length to user memory

Synopsis

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

Arguments

<i>addr</i>	The user address to write the string to
<i>n</i>	The maximum length of the string
<i>val</i>	The string which is to be written

Description

Writes the given string up to a maximum given length to a given user 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**).

function::mdelay

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.

function::panic

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.

function::raise

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.

function::udelay

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 searching, escaping, tokenizing, and converting strings to longs.

function::isdigit

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.

function::instr

function::instr — Returns whether a string is a substring of another string

Synopsis

```
instr: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.

function::matched

function::matched — Return a given matched subexpression.

Synopsis

```
matched:string(n:long)
```

Arguments

n index to the subexpression to return. 0 corresponds to the entire regular expression.

Description

returns the content of the *n*'th subexpression of the last successful use of the `=~` regex matching operator. Returns an empty string if the *n*'th subexpression was not matched (e.g. due to alternation). Throws an error if the last use of `=~` was a failed match, or if fewer than *n* subexpressions are present in the original regexp.

function::matched_str

function::matched_str — Return the last matched string.

Synopsis

```
matched_str:string()
```

Arguments

None

Description

returns the string matched by the last successful use of the `=~` regex matching operator. Returns an error if the last use of `=~` led to a failed match.

function::ngroups

function::ngroups — Number of subexpressions in the last match.

Synopsis

```
ngroups:long( )
```

Arguments

None

Description

returns the number of subexpressions from the last successful use of the `=~` regex matching operator.

Note that this number includes subexpressions which are present in the regex but did not match any string; for example, given the regex `"a|(b)"`, the subexpressions will count the group for (b) regardless of whether it matched a string or not. Throws an error if the last use of `=~` was a failed match.

function::str_replace

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

Synopsis

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

Arguments

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

Description

This function returns the given string with substrings replaced.

function::string_quoted

function::string_quoted — Quotes a given string

Synopsis

```
string_quoted:string(str:string)
```

Arguments

str The kernel address to retrieve the string from

Description

Returns the quoted string version of the given string, with 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.

function::stringat

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.

function::strlen

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.

function::strpos

function::strpos — Returns location of a substring within another string

Synopsis

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

Arguments

s1 string to search in

s2 substring to find

Description

This function returns location of the first occurrence of string *s2* within *s1*, namely the return value is 0 in case *s2* is a prefix of *s1*. If *s2* is not a substring of *s1*, then the return value is -1.

function::strtol

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).

function::substr

function::substr — Returns a substring

Synopsis

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

Arguments

<i>str</i>	the string to take a substring from
<i>start</i>	starting position of the extracted string (first character is 0)
<i>length</i>	length of string to return

Description

Returns the substring of the 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).

function::text_str

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.

function::text_strn

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

Synopsis

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

Arguments

<i>input</i>	the string to escape
<i>len</i>	maximum length of string to return (0 implies MAXSTRINGLEN)
<i>quoted</i>	put double quotes around the string. If input string is truncated it will have “...” after the second quote

Description

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

function::tokenize

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

Synopsis

- 1) `tokenize:string(delim:string)`
- 2) `tokenize:string(input:string,delim:string)`

Arguments

- delim* set of characters that delimit the tokens
- input* string to tokenize. If empty, returns the next non-empty token in the string passed in the previous call to `tokenize`.

Description

- 1) This function 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.
- 2) 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.

function::ansi_clear_screen

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.

function::ansi_cursor_hide

function::ansi_cursor_hide — Hides the cursor.

Synopsis

```
ansi_cursor_hide()
```

Arguments

None

Description

Sends ansi code for hiding the cursor.

function::ansi_cursor_move

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.

function::ansi_cursor_restore

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`.

function::ansi_cursor_save

function::ansi_cursor_save — Saves the cursor position.

Synopsis

```
ansi_cursor_save( )
```

Arguments

None

Description

Sends ansi code for saving the current cursor position.

function::ansi_cursor_show

function::ansi_cursor_show — Shows the cursor.

Synopsis

```
ansi_cursor_show( )
```

Arguments

None

Description

Sends ansi code for showing the cursor.

function::ansi_new_line

function::ansi_new_line — Move cursor to new line.

Synopsis

```
ansi_new_line()
```

Arguments

None

Description

Sends ansi code new line.

function::ansi_reset_color

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.

function::ansi_set_color

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

Synopsis

- 1) `ansi_set_color(fg:long)`
- 2) `ansi_set_color(fg:long,bg:long)`
- 3) `ansi_set_color(fg:long,bg:long,attr:long)`

Arguments

- fg* Foreground color to set.
- bg* Background color to set.
- attr* Color attribute to set.

Description

- 1) 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).
- 2) 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).
- 3) 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).

function::indent

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.

function::indent_depth

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.

function::thread_indent

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.

function::thread_indent_depth

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:

probe::stap.cache_add_mod

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

Synopsis

```
stap.cache_add_mod
```

Values

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

Description

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

probe::stap.cache_add_nss

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

Synopsis

```
stap.cache_add_nss
```

Values

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

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

Description

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

probe::stap.cache_add_src

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

Synopsis

```
stap.cache_add_src
```

Values

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

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

Description

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

probe::stap.cache_clean

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.

probe::stap.cache_get

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.

probe::stap.pass0

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.

probe::stap.pass0.end

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.

probe::stap.pass1.end

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.

probe::stap.pass1a

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.

probe::stap.pass1b

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.

probe::stap.pass2

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`.

probe::stap.pass2.end

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

probe::stap.pass3

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`.

probe::stap.pass3.end

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

probe::stap.pass4

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`.

probe::stap.pass4.end

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

probe::stap.pass5

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`.

probe::stap.pass5.end

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

probe::stap.pass6

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

probe::stap.pass6.end

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.

probe::stap.system

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.

probe::stap.system.return

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`.

probe::stap.system.spawn

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`.

probe::stapio.receive_control_message

probe::stapio.receive_control_message — Received a control message

Synopsis

```
stapio.receive_control_message
```

Values

len the length (in bytes) of the data blob

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

Description

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

probe::staprun.insert_module

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.

probe::staprun.remove_module

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.

probe::staprun.send_control_message

probe::staprun.send_control_message — Sending a control message

Synopsis

```
staprun.send_control_message
```

Values

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

len the length (in bytes) of the data blob

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

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.

function::nfsderror

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.

probe::nfs.aop.readpage

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

Synopsis

`nfs.aop.readpage`

Values

<i>file</i>	file argument
<i>sb_flag</i>	super block flags
<i>rsize</i>	read size (in bytes)
<i>size</i>	number of pages to be read in this execution
<i>ino</i>	inode number
<i>page_index</i>	offset within mapping, can used a page identifier and position identifier in the page frame
<i>i_flag</i>	file flags
<i>__page</i>	the address of page
<i>i_size</i>	file length in bytes
<i>dev</i>	device identifier

Description

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

probe::nfs.aop.readpages

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

Synopsis

`nfs.aop.readpages`

Values

<i>file</i>	filp argument
<i>size</i>	number of pages attempted to read in this execution
<i>rsize</i>	read size (in bytes)
<i>rpages</i>	read size (in pages)
<i>nr_pages</i>	number of pages attempted to read in this execution
<i>ino</i>	inode number
<i>dev</i>	device identifier

Description

Fires when in readahead way, read several pages once

probe::nfs.aop.release_page

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

Synopsis

`nfs.aop.release_page`

Values

<i>dev</i>	device identifier
<i>__page</i>	the address of page
<i>page_index</i>	offset within mapping, can used a page identifier and position identifier in the page frame
<i>ino</i>	inode number
<i>size</i>	release pages

Description

Fires when do a release operation on NFS.

probe::nfs.aop.set_page_dirty

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

Synopsis

```
nfs.aop.set_page_dirty
```

Values

__page the address of page

page_flag page flags

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.

probe::nfs.aop.write_begin

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

Synopsis

```
nfs.aop.write_begin
```

Values

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

probe::nfs.aop.write_end

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

Description

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

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

probe::nfs.aop.writepage

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

Synopsis

nfs.aop.writepage

Values

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

Description

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

probe::nfs.aop.writepages

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

Synopsis

```
nfs.aop.writepages
```

Values

<i>ino</i>	inode number
<i>wsiz</i> e	write size
<i>dev</i>	device identifier
<i>wpag</i> es	write size (in pages)
<i>for_kup</i> date	a flag of writeback_control, indicates if it's a kupdate writeback
<i>for_re</i> claim	a flag of writeback_control, indicates if it's invoked from the page allocator
<i>nr_to</i> _write	number of pages attempted to be written in this execution
<i>size</i>	number of pages attempted to be written in this execution

Description

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

probe::nfs.fop.aio_read

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

Synopsis

`nfs.fop.aio_read`

Values

<i>parent_name</i>	parent dir name
<i>count</i>	read bytes
<i>dev</i>	device identifier
<i>file_name</i>	file name
<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>ino</i>	inode number
<i>cache_valid</i>	cache related bit mask flag
<i>pos</i>	current position of file
<i>cache_time</i>	when we started read-caching this inode
<i>buf</i>	the address of buf in user space

probe::nfs.fop.aio_write

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

Synopsis

```
nfs.fop.aio_write
```

Values

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

probe::nfs.fop.check_flags

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

Synopsis

`nfs.fop.check_flags`

Values

flag file flag

probe::nfs.fop.flush

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

Synopsis

```
nfs.fop.flush
```

Values

<i>ndirty</i>	number of dirty page
<i>mode</i>	file mode
<i>ino</i>	inode number
<i>dev</i>	device identifier

probe::nfs.fop.fsync

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

Synopsis

`nfs.fop.fsync`

Values

<i>dev</i>	device identifier
<i>ino</i>	inode number
<i>ndirty</i>	number of dirty pages

probe::nfs.fop.llseek

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

Synopsis

`nfs.fop.llseek`

Values

<i>ino</i>	inode number
<i>whence</i>	the position to seek from
<i>whence_str</i>	symbolic string representation of the position to seek from
<i>offset</i>	the offset of the file will be repositioned
<i>dev</i>	device identifier

probe::nfs.fop.lock

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

Synopsis

`nfs.fop.lock`

Values

<i>fl_end</i>	ending offset of locked region
<i>fl_flag</i>	lock flags
<i>fl_type</i>	lock type
<i>fl_start</i>	starting offset of locked region
<i>dev</i>	device identifier
<i>cmd</i>	cmd arguments
<i>ino</i>	inode number
<i>i_mode</i>	file type and access rights

probe::nfs.fop.mmap

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

Synopsis

`nfs.fop.mmap`

Values

<i>vm_flag</i>	vm flags
<i>buf</i>	the address of buf in user space
<i>vm_start</i>	start address within vm_mm
<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>ino</i>	inode number
<i>cache_valid</i>	cache related bit mask flag
<i>parent_name</i>	parent dir name
<i>file_name</i>	file name
<i>vm_end</i>	the first byte after end address within vm_mm
<i>dev</i>	device identifier

probe::nfs.fop.open

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

Synopsis

`nfs.fop.open`

Values

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

probe::nfs.fop.read

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.

probe::nfs.fop.read_iter

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

Synopsis

`nfs.fop.read_iter`

Values

<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>pos</i>	current position of file
<i>cache_valid</i>	cache related bit mask flag
<i>ino</i>	inode number
<i>parent_name</i>	parent dir name
<i>file_name</i>	file name
<i>count</i>	read bytes
<i>dev</i>	device identifier
<i>cache_time</i>	when we started read-caching this inode

probe::nfs.fop.release

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

Synopsis

```
nfs.fop.release
```

Values

<i>mode</i>	file mode
<i>ino</i>	inode number
<i>dev</i>	device identifier

probe::nfs.fop.sendfile

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>ino</i>	inode number
<i>cache_valid</i>	cache related bit mask flag
<i>dev</i>	device identifier
<i>count</i>	read bytes

probe::nfs.fop.write

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.

probe::nfs.fop.write_iter

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

Synopsis

```
nfs.fop.write_iter
```

Values

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

probe::nfs.proc.commit

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

Synopsis

`nfs.proc.commit`

Values

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

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.

probe::nfs.proc.commit_done

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

Synopsis

nfs.proc.commit_done

Values

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

Description

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

probe::nfs.proc.commit_setup

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

Synopsis

nfs.proc.commit_setup

Values

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

Description

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

probe::nfs.proc.create

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

Synopsis

`nfs.proc.create`

Values

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

probe::nfs.proc.handle_exception

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.

probe::nfs.proc.lookup

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

Synopsis

`nfs.proc.lookup`

Values

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

probe::nfs.proc.open

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

Synopsis

```
nfs.proc.open
```

Values

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

Description

Allocate file read/write context information

probe::nfs.proc.read

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

Synopsis

`nfs.proc.read`

Values

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

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.

probe::nfs.proc.read_done

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>timestamp</i>	V4 timestamp, which is used for lease renewal
<i>version</i>	NFS version
<i>status</i>	result of last operation
<i>prot</i>	transfer protocol

Description

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

probe::nfs.proc.read_setup

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

Synopsis

```
nfs.proc.read_setup
```

Values

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

Description

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

probe::nfs.proc.release

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

Synopsis

```
nfs.proc.release
```

Values

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

Description

Release file read/write context information

probe::nfs.proc.remove

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

Synopsis

```
nfs.proc.remove
```

Values

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

probe::nfs.proc.rename

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

Synopsis

`nfs.proc.rename`

Values

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

probe::nfs.proc.rename_done

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

Synopsis

nfs.proc.rename_done

Values

<i>timestamp</i>	V4 timestamp, which is used for lease renewal
<i>old_fh</i>	file handle of old parent dir
<i>new_fh</i>	file handle of new parent dir
<i>server_ip</i>	IP address of server
<i>version</i>	NFS version
<i>status</i>	result of last operation
<i>prot</i>	transfer protocol

Description

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

probe::nfs.proc.rename_setup

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

Synopsis

```
nfs.proc.rename_setup
```

Values

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

probe::nfs.proc.write

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

Synopsis

```
nfs.proc.write
```

Values

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

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.

probe::nfs.proc.write_done

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

Synopsis

```
nfs.proc.write_done
```

Values

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

Description

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

probe::nfs.proc.write_setup

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

Synopsis

```
nfs.proc.write_setup
```

Values

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

Description

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

probe::nfsd.close

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

Synopsis

```
nfsd.close
```

Values

<i>filename</i>	file name
-----------------	-----------

Description

This probe point does not exist in kernels starting with 4.2.

probe::nfsd.commit

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

Synopsis

`nfsd.commit`

Values

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

probe::nfsd.create

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

Synopsis

`nfsd.create`

Values

<i>client_ip</i>	the ip address of client
<i>fh</i>	file handle (the first part is the length of the file handle)
<i>iap_mode</i>	file access mode
<i>type</i>	file type(regular,dir,device,fifo ...)
<i>iap_valid</i>	Attribute flags
<i>filelen</i>	the length of file name
<i>filename</i>	file name

Description

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

probe::nfsd.createv3

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

Description

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

probe::nfsd.dispatch

probe::nfsd.dispatch — NFS server receives an operation from client

Synopsis

`nfsd.dispatch`

Values

<i>client_ip</i>	the ip address of client
<i>proto</i>	transfer protocol
<i>xid</i>	transmission id
<i>prog</i>	program number
<i>version</i>	nfs version
<i>proc</i>	procedure number

probe::nfsd.lookup

probe::nfsd.lookup — NFS server opening or searching file for a file for client

Synopsis

`nfsd.lookup`

Values

<i>filename</i>	file name
<i>filelen</i>	the length of file name
<i>fh</i>	file handle of parent dir(the first part is the length of the file handle)
<i>client_ip</i>	the ip address of client

probe::nfsd.open

probe::nfsd.open — NFS server opening a file for client

Synopsis

`nfsd.open`

Values

<i>access</i>	indicates the type of open (read/write/commit/readdir...)
<i>client_ip</i>	the ip address of client
<i>fh</i>	file handle (the first part is the length of the file handle)
<i>type</i>	type of file (regular file or dir)

probe::nfsd.proc.commit

probe::nfsd.proc.commit — NFS server performing a commit operation for client

Synopsis

`nfsd.proc.commit`

Values

<i>offset</i>	the offset of file
<i>fh</i>	file handle (the first part is the length of the file handle)
<i>proto</i>	transfer protocol
<i>client_ip</i>	the ip address of client
<i>gid</i>	requester's group id
<i>size</i>	read bytes
<i>uid</i>	requester's user id
<i>version</i>	nfs version
<i>count</i>	read bytes

probe::nfsd.proc.create

probe::nfsd.proc.create — NFS server creating a file for client

Synopsis

`nfsd.proc.create`

Values

<i>filelen</i>	length of file name
<i>gid</i>	requester's group id
<i>filename</i>	file name
<i>uid</i>	requester's user id
<i>version</i>	nfs version
<i>client_ip</i>	the ip address of client
<i>proto</i>	transfer protocol
<i>fh</i>	file handle (the first part is the length of the file handle)

probe::nfsd.proc.lookup

probe::nfsd.proc.lookup — NFS server opening or searching for a file for client

Synopsis

`nfsd.proc.lookup`

Values

<i>version</i>	nfs version
<i>uid</i>	requester's user id
<i>filename</i>	file name
<i>gid</i>	requester's group id
<i>filelen</i>	the length of 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>client_ip</i>	the ip address of client

probe::nfsd.proc.read

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>uid</i>	requester's user id
<i>version</i>	nfs version
<i>vlen</i>	read blocks
<i>count</i>	read bytes
<i>fh</i>	file handle (the first part is the length of the file handle)
<i>offset</i>	the offset of file
<i>proto</i>	transfer protocol
<i>vec</i>	struct kvec, includes buf address in kernel address and length of each buffer
<i>client_ip</i>	the ip address of client

probe::nfsd.proc.remove

probe::nfsd.proc.remove — NFS server removing a file for client

Synopsis

`nfsd.proc.remove`

Values

<i>filename</i>	file name
<i>filelen</i>	length of file name
<i>gid</i>	requester's group id
<i>version</i>	nfs version
<i>uid</i>	requester's user id
<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

probe::nfsd.proc.rename

probe::nfsd.proc.rename — NFS Server renaming a file for client

Synopsis

`nfsd.proc.rename`

Values

<i>client_ip</i>	the ip address of client
<i>fh</i>	file handler of old path
<i>tfh</i>	file handler of new path
<i>tname</i>	new file name
<i>flen</i>	length of old file name
<i>tlen</i>	length of new file name
<i>gid</i>	requester's group id
<i>filename</i>	old file name
<i>uid</i>	requester's user id

probe::nfsd.proc.write

probe::nfsd.proc.write — NFS server writing data to file for client

Synopsis

`nfsd.proc.write`

Values

<i>stable</i>	argp->stable
<i>client_ip</i>	the ip address of client
<i>vec</i>	struct kvec, includes buf address in kernel address and length of each buffer
<i>proto</i>	transfer protocol
<i>offset</i>	the offset of file
<i>fh</i>	file handle (the first part is the length of the file handle)
<i>count</i>	read bytes
<i>vlen</i>	read blocks
<i>size</i>	read bytes
<i>uid</i>	requester's user id
<i>version</i>	nfs version
<i>gid</i>	requester's group id

probe::nfsd.read

probe::nfsd.read — NFS server reading data from a file for client

Synopsis

`nfsd.read`

Values

<i>size</i>	read bytes
<i>count</i>	read bytes
<i>file</i>	argument file, indicates if the file has been opened.
<i>vlen</i>	read blocks
<i>vec</i>	struct kvec, includes buf address in kernel address and length of each buffer
<i>offset</i>	the offset of file
<i>fh</i>	file handle (the first part is the length of the file handle)
<i>client_ip</i>	the ip address of client

probe::nfsd.rename

probe::nfsd.rename — NFS server renaming a file for client

Synopsis

`nfsd.rename`

Values

<i>fh</i>	file handler of old path
<i>tname</i>	new file name
<i>tfh</i>	file handler of new path
<i>client_ip</i>	the ip address of client
<i>filename</i>	old file name
<i>flen</i>	length of old file name
<i>tlen</i>	length of new file name

probe::nfsd.unlink

probe::nfsd.unlink — NFS server removing a file or a directory for client

Synopsis

```
nfsd.unlink
```

Values

<i>filename</i>	file name
<i>filelen</i>	the length of file name
<i>fh</i>	file handle (the first part is the length of the file handle)
<i>type</i>	file type (file or dir)
<i>client_ip</i>	the ip address of client

probe::nfsd.write

probe::nfsd.write — NFS server writing data to a file for client

Synopsis

```
nfsd.write
```

Values

<i>count</i>	read bytes
<i>file</i>	argument file, indicates if the file has been opened.
<i>vlen</i>	read blocks
<i>size</i>	read bytes
<i>client_ip</i>	the ip address of client
<i>vec</i>	struct kvec, includes buf address in kernel address and length of each buffer
<i>fh</i>	file handle (the first part is the length of the file handle)
<i>offset</i>	the offset of file

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.

function::commit

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*.

function::discard

function::discard — Discard all output related to a speculation buffer

Synopsis

```
discard(id:long)
```

Arguments

id of the buffer to store the information in

function::speculate

function::speculate — Store a string for possible output later

Synopsis

```
speculate(id:long,output:string)
```

Arguments

<i>id</i>	buffer id to store the information in
<i>output</i>	string to write out when commit occurs

Description

Add a string to the speculaive buffer for id.

function::speculation

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:

function::json_add_array

function::json_add_array — Add an array

Synopsis

```
json_add_array:long(name:string,description:string)
```

Arguments

<i>name</i>	The name of the array.
<i>description</i>	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`.

function::json_add_array_numeric_metric

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
```

Arguments

<i>array_name</i>	The name of the array the numeric metric should be added to.
<i>metric_name</i>	The name of the numeric metric.
<i>metric_description</i>	Metric description. An empty string can be used.
<i>metric_units</i>	Metic units. An empty string can be used.

Description

This function adds a numeric metric to an array, setting up everything needed.

function::json_add_array_string_metric

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_
```

Arguments

<i>array_name</i>	The name of the array the string metric should be added to.
<i>metric_name</i>	The name of the string metric.
<i>metric_description</i>	Metric description. An empty string can be used.

Description

This function adds a string metric to an array, setting up everything needed.

function::json_add_numeric_metric

function::json_add_numeric_metric — Add a numeric metric

Synopsis

```
json_add_numeric_metric:long(name:string,description:string,units:string)
```

Arguments

<i>name</i>	The name of the numeric metric.
<i>description</i>	Metric description. An empty string can be used.
<i>units</i>	Metic units. An empty string can be used.

Description

This function adds a numeric metric, setting up everything needed.

function::json_add_string_metric

function::json_add_string_metric — Add a string metric

Synopsis

```
json_add_string_metric:long(name:string,description:string)
```

Arguments

<i>name</i>	The name of the string metric.
<i>description</i>	Metric description. An empty string can be used.

Description

This function adds a string metric, setting up everything needed.

function::json_set_prefix

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.

macro::json_output_array_numeric_value

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

<i>array_name</i>	The name of the array.
<i>array_index</i>	The array index (as a string) indicating where to store the numeric value.
<i>metric_name</i>	The name of the numeric metric.
<i>value</i>	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`.

macro::json_output_array_string_value

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

<i>array_name</i>	The name of the array.
<i>array_index</i>	The array index (as a string) indicating where to store the string value.
<i>metric_name</i>	The name of the string metric.
<i>value</i>	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`.

macro::json_output_data_end

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.

macro::json_output_data_start

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.

macro::json_output_numeric_value

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`.

macro::json_output_string_value

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`.

probe::json_data

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.

Chapter 34. Output file switching Tapset

Utility function to allow switching of output files.

function::switch_file

function::switch_file — switch to the next output file

Synopsis

```
switch_file()
```

Arguments

None

Description

This function sends a signal to the stapio process, commanding it to rotate to the next output file when output is sent to file(s).