SystemTap Tapset Reference Manual

SystemTap
# Table of Contents

1. Introduction ........................................................................................................... 1
2. Context Functions .................................................................................................. 2
   function::addr ........................................................................................................ 3
   function::asmlinkage .............................................................................................. 4
   function::backtrace .............................................................................................. 5
   function::caller ...................................................................................................... 6
   function::caller_addr ............................................................................................ 7
   function::callers .................................................................................................... 8
   function::cmdline_arg ............................................................................................ 9
   function::cmdline_args ......................................................................................... 10
   function::cmdline_str ......................................................................................... 11
   function::cpu .......................................................................................................... 12
   function::cpuid ...................................................................................................... 13
   function::egid .......................................................................................................... 14
   function::env_var .................................................................................................. 15
   function::euid ......................................................................................................... 16
   function::execname ............................................................................................... 17
   function::fastcall ................................................................................................. 18
   function::gid .......................................................................................................... 19
   function::int_arg ................................................................................................... 20
   function::is_myproc .............................................................................................. 21
   function::is_return ............................................................................................... 22
   function::long_arg ................................................................................................. 23
   function::longlong_arg ......................................................................................... 24
   function::modname ............................................................................................... 25
   function::module_name ......................................................................................... 26
   function::ns_egid ................................................................................................... 27
   function::ns_euid .................................................................................................. 28
   function::ns_gid .................................................................................................... 29
   function::ns_pgrp ................................................................................................. 30
   function::ns_pid .................................................................................................... 31
   function::ns_ppid .................................................................................................. 32
   function::ns_sid ..................................................................................................... 33
   function::ns_tid ...................................................................................................... 34
   function::ns_uid .................................................................................................... 35
   function::pexecname ............................................................................................. 36
   function::pgrp ........................................................................................................ 37
   function::pid .......................................................................................................... 38
   function::pid2execname ....................................................................................... 39
   function::pid2task ................................................................................................. 40
   function::pn ............................................................................................................ 41
   function::pnlabel ................................................................................................. 42
   function::pointer_arg ........................................................................................... 43
   function::pp ........................................................................................................... 44
   function::ppfunc ................................................................................................... 45
   function::ppid ........................................................................................................ 46
   function::print_backtrace ..................................................................................... 47
   function::print_regs ............................................................................................. 48
   function::print_stack ............................................................................................ 49
   function::print_syms ............................................................................................ 50
   function::print_ubacktrace .................................................................................. 51
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>print_ubacktrace_brief</code></td>
<td>52</td>
</tr>
<tr>
<td><code>print_ustack</code></td>
<td>53</td>
</tr>
<tr>
<td><code>print_usyms</code></td>
<td>54</td>
</tr>
<tr>
<td><code>probe_type</code></td>
<td>55</td>
</tr>
<tr>
<td><code>probfunc</code></td>
<td>56</td>
</tr>
<tr>
<td><code>probemod</code></td>
<td>57</td>
</tr>
<tr>
<td><code>pstrace</code></td>
<td>58</td>
</tr>
<tr>
<td><code>register</code></td>
<td>59</td>
</tr>
<tr>
<td><code>registers_valid</code></td>
<td>60</td>
</tr>
<tr>
<td><code>regparm</code></td>
<td>61</td>
</tr>
<tr>
<td><code>remote_id</code></td>
<td>62</td>
</tr>
<tr>
<td><code>remote_uri</code></td>
<td>63</td>
</tr>
<tr>
<td><code>s32_arg</code></td>
<td>64</td>
</tr>
<tr>
<td><code>s64_arg</code></td>
<td>65</td>
</tr>
<tr>
<td><code>sid</code></td>
<td>66</td>
</tr>
<tr>
<td><code>sprint_backtrace</code></td>
<td>67</td>
</tr>
<tr>
<td><code>sprint_stack</code></td>
<td>68</td>
</tr>
<tr>
<td><code>sprint_syms</code></td>
<td>69</td>
</tr>
<tr>
<td><code>sprint_ubacktrace</code></td>
<td>70</td>
</tr>
<tr>
<td><code>sprint_ustack</code></td>
<td>71</td>
</tr>
<tr>
<td><code>sprint_usyms</code></td>
<td>72</td>
</tr>
<tr>
<td><code>stack</code></td>
<td>73</td>
</tr>
<tr>
<td><code>stack_size</code></td>
<td>74</td>
</tr>
<tr>
<td><code>stack_unused</code></td>
<td>75</td>
</tr>
<tr>
<td><code>stack_used</code></td>
<td>76</td>
</tr>
<tr>
<td><code>stp_pid</code></td>
<td>77</td>
</tr>
<tr>
<td><code>symdata</code></td>
<td>78</td>
</tr>
<tr>
<td><code>symfile</code></td>
<td>79</td>
</tr>
<tr>
<td><code>symfileline</code></td>
<td>80</td>
</tr>
<tr>
<td><code>symline</code></td>
<td>81</td>
</tr>
<tr>
<td><code>symname</code></td>
<td>82</td>
</tr>
<tr>
<td><code>target</code></td>
<td>83</td>
</tr>
<tr>
<td><code>task_ancestry</code></td>
<td>84</td>
</tr>
<tr>
<td><code>task_backtrace</code></td>
<td>85</td>
</tr>
<tr>
<td><code>task_cpu</code></td>
<td>86</td>
</tr>
<tr>
<td><code>task_current</code></td>
<td>87</td>
</tr>
<tr>
<td><code>task_egid</code></td>
<td>88</td>
</tr>
<tr>
<td><code>task_euid</code></td>
<td>89</td>
</tr>
<tr>
<td><code>task_execname</code></td>
<td>90</td>
</tr>
<tr>
<td><code>task_fd_lookup</code></td>
<td>91</td>
</tr>
<tr>
<td><code>task_gid</code></td>
<td>92</td>
</tr>
<tr>
<td><code>task_max_file_handles</code></td>
<td>93</td>
</tr>
<tr>
<td><code>task_nice</code></td>
<td>94</td>
</tr>
<tr>
<td><code>task_ns_egid</code></td>
<td>95</td>
</tr>
<tr>
<td><code>task_ns_euid</code></td>
<td>96</td>
</tr>
<tr>
<td><code>task_ns_gid</code></td>
<td>97</td>
</tr>
<tr>
<td><code>task_ns_pid</code></td>
<td>98</td>
</tr>
<tr>
<td><code>task_ns_tid</code></td>
<td>99</td>
</tr>
<tr>
<td><code>task_ns_uid</code></td>
<td>100</td>
</tr>
<tr>
<td><code>task_open_file_handles</code></td>
<td>101</td>
</tr>
<tr>
<td><code>task_parent</code></td>
<td>102</td>
</tr>
<tr>
<td><code>task_pid</code></td>
<td>103</td>
</tr>
<tr>
<td><code>task_prio</code></td>
<td>104</td>
</tr>
<tr>
<td><code>task_state</code></td>
<td>105</td>
</tr>
</tbody>
</table>
function::task_tid .......................................................... 106
function::task_uid .......................................................... 107
function::tid .................................................................. 108
function::u32_arg ................................................................ 109
function::u64_arg ................................................................ 110
function::u_register ............................................................. 111
function::uaddr .................................................................. 112
function::ubacktrace ............................................................ 113
function::ucallers ................................................................ 114
function::uid .................................................................... 115
function::uint_arg ............................................................... 116
function::ulong_arg ............................................................. 117
function::ulonglong_arg ......................................................... 118
function::umodname .............................................................. 119
function::user_mode ............................................................. 120
function::ustack .................................................................. 121
function::usymdata .............................................................. 122
function::usymfile ............................................................... 123
function::usymfileline .......................................................... 124
function::usymline ............................................................... 125
function::usymname .............................................................. 126

3. Timestamp Functions .......................................................... 127
function::HZ ........................................................................ 128
function::cpu_clock_ms ......................................................... 129
function::cpu_clock_ns ......................................................... 130
function::cpu_clock_s .......................................................... 131
function::cpu_clock_us ........................................................ 132
function::delete_stopwatch .................................................... 133
function::get_cycles ............................................................ 134
function::gettimeofday_ms .................................................... 135
function::gettimeofday_ns ..................................................... 136
function::gettimeofday_s ....................................................... 137
function::jiffies ................................................................ 138
function::local_clock_ms ....................................................... 140
function::local_clock_ns ....................................................... 141
function::local_clock_s ........................................................ 142
function::local_clock_us ....................................................... 143
function::read_stopwatch_ms .................................................. 144
function::read_stopwatch_ns .................................................. 145
function::read_stopwatch_s .................................................... 146
function::read_stopwatch_us .................................................. 147
function::start_stopwatch ..................................................... 148
function::stop_stopwatch ...................................................... 149

4. Time utility functions .......................................................... 150
function::ctime ................................................................ 151
function::tz_ctime .............................................................. 152
function::tz_gmtoff ............................................................. 153
function::tz_name ............................................................... 154

5. Shell command functions ..................................................... 155
function::system ................................................................. 156

6. Memory Tapset ................................................................. 157
function::addr_to_node ........................................................ 158
function::bytes_to_string ...................................................... 159
function::mem_page_size .......................................................... 160
function::pages_to_string ......................................................... 161
function::proc_mem_data .......................................................... 162
function::proc_mem_data_pid ...................................................... 163
function::proc_mem_rss ........................................................... 164
function::proc_mem_rss_pid ....................................................... 165
function::proc_mem_shr .......................................................... 166
function::proc_mem_shr_pid ...................................................... 167
function::proc_mem_size ......................................................... 168
function::proc_mem_size_pid ..................................................... 169
function::proc_mem_string ....................................................... 170
function::proc_mem_string_pid .................................................. 171
function::proc_mem_txt .......................................................... 172
function::proc_mem_txt_pid ...................................................... 173
function::vm_faultContains .................................................... 174
probe::vm.brk ................................................................. 175
probe::vm.kfree ............................................................... 176
probe::vm.kmalloc ............................................................. 177
probe::vm.kmalloc_node .......................................................... 178
probe::vm.kmem_cache_alloc .................................................... 179
probe::vm.kmem_cache_alloc_node .............................................. 180
probe::vm.kmem_cache_free ..................................................... 181
probe::vm.mmap ................................................................. 182
probe::vm.munmap ............................................................... 183
probe::vm.oom_kill ............................................................. 184
probe::vm.pagefault ............................................................. 185
probe::vm.pagefault.return ..................................................... 186
probe::vm.write_shared .......................................................... 187
probe::vm.write_shared_copy ................................................... 188
7. Task Time Tapset ................................................................. 189
function::cputime_to_msecs ..................................................... 190
function::cputime_to_string .................................................... 191
function::cputime_to_usecs ..................................................... 192
function::msecs_to_string ....................................................... 193
function::nsecs_to_string ....................................................... 194
function::nsecs_to_usecs ....................................................... 195
function::task_start_time ....................................................... 196
function::task_stime .......................................................... 197
function::task_stime_tid ........................................................ 198
function::task_time ............................................................ 199
function::task_time_string ..................................................... 200
function::task_time_string_tid ................................................ 201
function::usecs_to_string ...................................................... 202
8. Scheduler Tapset ................................................................. 203
probe::scheduler.balance ......................................................... 204
probe::scheduler.cpu_off ........................................................ 205
probe::scheduler.cpu_on .......................................................... 206
probe::scheduler.ctctxswitch ................................................... 207
probe::scheduler.kthread_stop .................................................. 208
probe::scheduler.kthread_stop.return ....................................... 209
probe::scheduler.migrate ........................................................ 210
probe::scheduler.process_exit .................................................. 211
probe::scheduler.process_fork ................................................ 212
probe::scheduler.process_free .................................................. 213
<table>
<thead>
<tr>
<th>Section</th>
<th>Tapset Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>IO Scheduler and block IO Tapset</td>
</tr>
<tr>
<td></td>
<td>probe::scheduler.process_wait</td>
</tr>
<tr>
<td></td>
<td>probe::scheduler.signal_send</td>
</tr>
<tr>
<td></td>
<td>probe::scheduler.tick</td>
</tr>
<tr>
<td></td>
<td>probe::scheduler.wait_task</td>
</tr>
<tr>
<td></td>
<td>probe::scheduler.wakeup</td>
</tr>
<tr>
<td></td>
<td>probe::scheduler.wakeup_new</td>
</tr>
<tr>
<td>10</td>
<td>SCSI Tapset</td>
</tr>
<tr>
<td></td>
<td>probe::scsi.iocompleted</td>
</tr>
<tr>
<td></td>
<td>probe::scsi.iodispatching</td>
</tr>
<tr>
<td></td>
<td>probe::scsi.iodone</td>
</tr>
<tr>
<td></td>
<td>probe::scsi.ioentry</td>
</tr>
<tr>
<td></td>
<td>probe::scsi.ioreexecute</td>
</tr>
<tr>
<td></td>
<td>probe::scsi.set_state</td>
</tr>
<tr>
<td>11</td>
<td>TTY Tapset</td>
</tr>
<tr>
<td></td>
<td>probe::tty.init</td>
</tr>
<tr>
<td></td>
<td>probe::tty.ioctl</td>
</tr>
<tr>
<td></td>
<td>probe::tty.open</td>
</tr>
<tr>
<td></td>
<td>probe::tty.poll</td>
</tr>
<tr>
<td></td>
<td>probe::tty.read</td>
</tr>
<tr>
<td></td>
<td>probe::tty.receive</td>
</tr>
<tr>
<td></td>
<td>probe::tty.register</td>
</tr>
<tr>
<td></td>
<td>probe::tty.release</td>
</tr>
<tr>
<td></td>
<td>probe::tty.resize</td>
</tr>
<tr>
<td></td>
<td>probe::tty.unregister</td>
</tr>
<tr>
<td></td>
<td>probe::tty.write</td>
</tr>
<tr>
<td>12</td>
<td>Interrupt Request (IRQ) Tapset</td>
</tr>
<tr>
<td></td>
<td>probe::irq_handler.entry</td>
</tr>
<tr>
<td></td>
<td>probe::irq_handler.exit</td>
</tr>
<tr>
<td></td>
<td>probe::softirq.entry</td>
</tr>
<tr>
<td></td>
<td>probe::softirq.exit</td>
</tr>
<tr>
<td></td>
<td>probe::workqueue.create</td>
</tr>
<tr>
<td></td>
<td>probe::workqueue.destroy</td>
</tr>
<tr>
<td></td>
<td>probe::workqueue.execute</td>
</tr>
<tr>
<td></td>
<td>probe::workqueue.insert</td>
</tr>
<tr>
<td>13</td>
<td>Networking Tapset</td>
</tr>
<tr>
<td></td>
<td>probe::network.tapset</td>
</tr>
<tr>
<td></td>
<td>probe::network.tapset</td>
</tr>
<tr>
<td></td>
<td>probe::network.tapset</td>
</tr>
<tr>
<td></td>
<td>probe::network.tapset</td>
</tr>
<tr>
<td></td>
<td>probe::network.tapset</td>
</tr>
<tr>
<td></td>
<td>probe::network.tapset</td>
</tr>
<tr>
<td></td>
<td>probe::network.tapset</td>
</tr>
<tr>
<td></td>
<td>probe::network.tapset</td>
</tr>
<tr>
<td></td>
<td>probe::network.tapset</td>
</tr>
<tr>
<td></td>
<td>probe::network.tapset</td>
</tr>
<tr>
<td></td>
<td>probe::network.tapset</td>
</tr>
<tr>
<td></td>
<td>probe::network.tapset</td>
</tr>
<tr>
<td></td>
<td>probe::network.tapset</td>
</tr>
<tr>
<td></td>
<td>probe::network.tapset</td>
</tr>
<tr>
<td></td>
<td>probe::network.tapset</td>
</tr>
<tr>
<td></td>
<td>probe::network.tapset</td>
</tr>
<tr>
<td></td>
<td>probe::network.tapset</td>
</tr>
<tr>
<td></td>
<td>probe::network.tapset</td>
</tr>
<tr>
<td></td>
<td>probe::network.tapset</td>
</tr>
<tr>
<td></td>
<td>probe::network.tapset</td>
</tr>
<tr>
<td></td>
<td>probe::network.tapset</td>
</tr>
<tr>
<td></td>
<td>probe::network.tapset</td>
</tr>
<tr>
<td></td>
<td>probe::network.tapset</td>
</tr>
</tbody>
</table>
14. Socket Tapset ................................................................. 346
   function::inet_get_ip_source ....................................... 347
   function::inet_get_local_port ..................................... 348
   function::sock_fam_num2str ....................................... 349
   function::sock_fam_str2num ....................................... 350
   function::sock_prot_num2str ..................................... 351
   function::sock_prot_str2num ..................................... 352
   function::sock_state_num2str ................................... 353
   function::socket.aio_read ......................................... 354
   function::socket.aio_read.return ................................ 355
   function::socket.aio_write ....................................... 356
   function::socket.aio_write.return ................................ 357
   function::socket.close ........................................... 358
   function::socket.close.return ................................... 359
   function::socket.create .......................................... 360
   function::socket.create.return .................................. 361
   function::socket.read ............................................ 362
   function::socket.read.iter ....................................... 363
   function::socket.read.iter.return ................................ 364
   function::socket.ready ........................................... 365
   function::socket.ready.return ................................... 366
   function::socket.receive ......................................... 367
   function::socket.recvmsg ......................................... 368
   function::socket.recvmsg.return ................................ 369
   function::socket.send ............................................. 370
   function::socket.sendmsg ......................................... 371
   function::socket.sendmsg.return ................................ 372
   function::socket.write ............................................ 373
   function::socket.write.iter ....................................... 374
   function::socket.write.iter.return ................................ 375
   function::socket.writev .......................................... 376
   function::socket.writev.return .................................. 377

15. SNMP Information Tapset .............................................. 377
   function::ipmib_filter_key ...................................... 378
   function::ipmib_get_proto ....................................... 379
   function::ipmib_local_addr ...................................... 380
   function::ipmib_remote_addr ..................................... 381
   function::ipmib_tcp_local_port ................................ 382
   function::ipmib_tcp_remote_port ................................ 383
   function::linuxmib_filter_key .................................. 384
   function::tcpmib_filter_key ................................... 385
function::tcpmib_get_state .......................................................... 386
function::tcpmib_local_addr .......................................................... 387
function::tcpmib_local_port .......................................................... 388
function::tcpmib_remote_addr ........................................................ 389
function::tcpmib_remote_port ......................................................... 390
probe::ipmib.ForwDatagrams .......................................................... 391
probe::ipmib.FragFails ................................................................. 392
probe::ipmib.FragOKs ................................................................. 393
probe::ipmib.InAddrErrors ............................................................ 394
probe::ipmib.InDiscards ............................................................... 395
probe::ipmib.InNoRoutes ............................................................... 396
probe::ipmib.InReceives ............................................................... 397
probe::ipmib.InUnknownProtos ...................................................... 398
probe::ipmib.OutRequests ............................................................. 399
probe::ipmib.ReasmReqs ............................................................... 400
probe::ipmib.ReasnTimeout ........................................................... 401
probe::linuxmib.DelayedACKs ......................................................... 402
probe::linuxmib.ListenDrops ........................................................ 403
probe::linuxmib.ListenOverflows .................................................. 404
probe::linuxmib.TCPMemoryPressures ............................................. 405
probe::tcpmib.ActiveOpens ........................................................... 406
probe::tcpmib.AttemptFails .......................................................... 407
probe::tcpmib.CurrEstab .............................................................. 408
probe::tcpmib.EstabResets ........................................................... 409
probe::tcpmib.InSegs ................................................................. 410
probe::tcpmib.OutSegs ............................................................... 411
probe::tcpmib.PassiveOpens ........................................................ 412
probe::tcpmib.RetransSegs ............................................................. 413
16. Kernel Process Tapset ............................................................... 415
   function::get_loadavg_index ....................................................... 416
   function::sprint_loadavg ........................................................... 417
   function::target_set_pid .......................................................... 418
   function::target_set_report ...................................................... 419
   probe::kprocess.create ............................................................ 420
   probe::kprocess.exec .............................................................. 421
   probe::kprocess.exec_complete .................................................. 422
   probe::kprocess.exit ............................................................... 423
   probe::kprocess.release .......................................................... 424
   probe::kprocess.start ............................................................. 425
17. Signal Tapset ........................................................................... 426
   function::get_sa_flags ............................................................... 427
   function::get_sa_handler ........................................................... 428
   function::is Sig_blockEd ............................................................ 429
   function::sa_flags_str .............................................................. 430
   function::sa_handler_str ........................................................... 431
   function::signal_str ............................................................... 432
   function::sigset_mask_str .......................................................... 433
   probe::signal.check_ignored ....................................................... 434
   probe::signal.check_ignored.return ............................................. 435
   probe::signal.checkperm ........................................................... 436
   probe::signal.checkperm.return .................................................. 437
   probe::signal.do_action ............................................................ 438
   probe::signal.do_action.return .................................................. 439
<table>
<thead>
<tr>
<th>Section</th>
<th>Function/Probes</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>18. Errno Tapset</td>
<td>function::errno_str, function::return_str, function::returnstr, function::returnval</td>
<td>460</td>
</tr>
<tr>
<td>19. RLIMIT Tapset</td>
<td>function::rlimit_from_str</td>
<td>465</td>
</tr>
<tr>
<td>20. Device Tapset</td>
<td>function::MAJOR, function::MINOR, function::MKDEV</td>
<td>467</td>
</tr>
<tr>
<td></td>
<td>function::usrdev2kerndev</td>
<td>470</td>
</tr>
<tr>
<td>21. Directory-entry (dentry) Tapset</td>
<td>function::d_name, function::d_path, function::fullpath_struct_file, function::fullpath_struct_nameidata, function::fullpath_struct_path</td>
<td>472</td>
</tr>
<tr>
<td></td>
<td>function::inode_name, function::inode_path, function::real_mount, function::reverse_path_walk, function::task_dentry_path</td>
<td>478</td>
</tr>
<tr>
<td>22. Logging Tapset</td>
<td>function::assert, function::error, function::exit, function::ftrace, function::log, function::printk, function::warn</td>
<td>483</td>
</tr>
<tr>
<td></td>
<td>function::assert, function::error, function::exit, function::ftrace, function::log, function::printk, function::warn</td>
<td>488</td>
</tr>
<tr>
<td>23. Queue Statistics Tapset</td>
<td>function::qs_done, function::qs_run</td>
<td>491</td>
</tr>
</tbody>
</table>

For more details, please refer to the SystemTap Tapset Reference Manual.
24. Random functions Tapset ................................................................. 493
function::random ................................................................. 494
function::random_uniform ...................................................... 495
function::random_normal ....................................................... 496
function::random_exponential .................................................. 497
function::random_lognormal .................................................... 498
function::random_zipf ............................................................ 499
function::random_beta ............................................................. 500
function::random_uniform_int .................................................. 501
function::random_normal_int .................................................... 502

25. String and data retrieving functions Tapset ......................................... 503
function::atomic_long_read .................................................... 504
function::atomic_read ............................................................ 505
function::atomic_long ............................................................. 506
function::atomic ................................................................. 507
function::kernel_char ............................................................. 508
function::kernel_int .............................................................. 509
function::kernel_long ............................................................ 510
function::kernel_pointer ....................................................... 511
function::kernel_short .......................................................... 512
function::kernel_string .......................................................... 513
function::kernel_string2 .......................................................... 514
function::kernel_string2_utf16 .................................................. 515
function::kernel_string2_utf32 .................................................. 516
function::kernel_string_n .......................................................... 517
function::kernel_string_quoted .................................................. 518
function::kernel_string_utf16 .................................................. 519
function::kernel_string_utf32 .................................................. 520
function::user_char .............................................................. 521
function::user_long ............................................................... 522
function::user_long_warn ....................................................... 523
function::user_int .............................................................. 524
function::user_int16 .............................................................. 525
function::user_int32 .............................................................. 526
function::user_int64 .............................................................. 527
function::user_int8 .............................................................. 528
function::user_int Warn ........................................................... 529
function::user_long ............................................................... 530
function::user_long_warn ....................................................... 531
function::user_short ............................................................. 532
function::user_short_warn ...................................................... 533
function::user_string ............................................................ 534
function::user_string2 ............................................................ 535
function::user_string2_n .......................................................... 536
function::user_string2_n2 .......................................................... 537
function::user_string2_n2_warn .................................................. 538
function::user_string2_utf16 .................................................... 539
function::user_string2_utf32 .................................................... 540
function::user_string2_utf32_warn .............................................. 541
function::user_string2_n .......................................................... 542
function::user_string2_n2 .......................................................... 543
function::user_string2_n2_warn .................................................. 544
function::user_string2_utf16 .................................................... 545
function::user_string2_utf32 .................................................... 546
function::user_string2_utf32_warn .............................................. 547
26. **String and data writing functions Tapset** .................................................. 556
function::set_kernel_char ................................................................. 557
function::set_kernel_int .................................................................. 558
function::set_kernel_long ............................................................... 559
function::set_kernel_pointer .......................................................... 560
function::set_kernel_short ............................................................. 561
function::set_kernel_string ............................................................ 562
function::set_kernel_string_n .......................................................... 563
27. **Guru tapsets** ...................................................................................... 564
function::mdelay .............................................................................. 565
function::panic ............................................................................... 566
function::raise ................................................................................ 567
function::udelay ............................................................................. 568
28. A collection of standard string functions .............................................. 569
function::isdigit ............................................................................ 570
function::isinst ............................................................................. 571
function::str_replace ...................................................................... 572
function::stringat ........................................................................ 573
function::strlen ............................................................................ 574
function::strtol ............................................................................. 575
function::substr .......................................................................... 576
function::text_str .......................................................................... 577
function::text_strn ......................................................................... 578
function::tokenize ......................................................................... 579
29. Utility functions for using ansi control chars in logs ................................ 580
function::ansi_clear_screen ............................................................ 581
function::ansi_cursor_hide ............................................................. 582
function::ansi_cursor_move ............................................................ 583
function::ansi_cursor_restore .......................................................... 584
function::ansi_cursor_save .............................................................. 585
function::ansi_cursor_show ............................................................. 586
function::ansi_new_line ................................................................. 587
function::ansi_reset_color .............................................................. 588
function::ansi_set_color ................................................................ 589
function::ansi_set_color2 .............................................................. 590
function::ansi_set_color3 .............................................................. 591
function::indent ............................................................................. 592
function::indent_depth ................................................................. 593
function::thread_indent ................................................................. 594
function::thread_indent_depth ....................................................... 595
30. **SystemTap Translator Tapset** ............................................................... 596
probe::stap.cache_add_mod .............................................................. 597
probe::stap.cache_add_nss ............................................................... 598
probe::stap.cache_add_src ............................................................... 599
probe::stap.cache_clean ................................................................. 600
probe::stap.cache_get ................................................................. 601
probe::stap.run.remove_module ............................................................... 622
probe::stap.run.send_control_message ................................................... 623

31. Network File Storage Tapsets ......................................................... 624
   function::nfserroror ................................................................. 625
   probe::nfs.aop.readpage ............................................................ 626
   probe::nfs.aop.readpages .......................................................... 627
   probe::nfs.aop.release_page ....................................................... 628
   probe::nfs.aop.set_page_dirty ..................................................... 629
   probe::nfs.aop.write_begin ....................................................... 630
   probe::nfs.aop.write_end .......................................................... 631
   probe::nfs.aop.writepage ........................................................... 632
   probe::nfs.aop.writepages .......................................................... 633
   probe::nfs.fop.aio_read ............................................................ 634
   probe::nfs.fop.aio_write ........................................................... 635
   probe::nfs.fop.check_flags ........................................................ 636
   probe::nfs.fop.flush ............................................................... 637
   probe::nfs.fop.fsync .............................................................. 638
   probe::nfs.fop.lseek .............................................................. 639
   probe::nfs.fop.lock ................................................................. 640
   probe::nfs.fop.mmap ............................................................... 641
   probe::nfs.fop.open .............................................................. 642
   probe::nfs.fop.read ............................................................... 643
   probe::nfs.fop.read_iter .......................................................... 644
   probe::nfs.fop.release ............................................................ 645
   probe::nfs.fop.sendfile ............................................................ 646
   probe::nfs.fop.write .............................................................. 647
   probe::nfs.fop.write_iter ........................................................ 648
   probe::nfs.proc.commit ............................................................. 649
   probe::nfs.proc.commit_done ...................................................... 650
   probe::nfs.proc.commit_setup ..................................................... 651
   probe::nfs.proc.create ........................................................... 652
   probe::nfs.proc.handle_exception ............................................... 653
   probe::nfs.proc.lookup ............................................................ 654
   probe::nfs.proc.open .............................................................. 655
32. Speculation ............................................................... 685
   function::commit .................................................. 686
   function::discard ................................................. 687
   function::speculate ............................................. 688
   function::speculation ........................................... 689
33. JSON Tapset ........................................................... 690
   function::json_add_array ........................................ 691
   function::json_add_array_numeric_metric ..................... 692
   function::json_add_array_string_metric ....................... 693
   function::json_add_numeric_metric ............................. 694
   function::json_add_string_metric .............................. 695
   function::json_set_prefix ....................................... 696
   macro::json_output_array_numeric_value ....................... 697
   macro::json_output_array_string_value ....................... 698
   macro::json_output_data_end ................................... 699
   macro::json_output_data_start .................................. 700
   macro::json_output_numeric_value ............................ 701
   macro::json_output_string_value ............................. 702
   probe::json_data .................................................. 703
34. Output file switching Tapset ...................................... 704
   function::switch_file ........................................... 705
35. syscalls ................................................................ 706
List of Tables

35.1. Syscalls list ........................................................................................................................................ 706
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 `syndata`. 
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\n0x\n", 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

    \n    \m    \n    \delim

    \n    \m    \n    \delim

    First argument to get (zero is normally the program itself)
    Last argument to get (or minus one for all arguments after n)
    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::egid

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

Synopsis

    \texttt{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

\texttt{env\_var:string(name:string)}

Arguments

\textit{name} \hspace{1em} 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

\[ \text{is}\_\text{return} : \text{long}() \]

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::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 process'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 process'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

\[ \text{print\_ubacktrace()} \]

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 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**

```c
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 if no symbol could be found).

**Note**

To get (full) backtraces for user space applications and shared libraries not mentioned in the current script run stap with `-d /path/to/exe-or-so` and/or add `--ldd` to load all needed unwind data.
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, rbx/bp, rbx/bx, rcx/cx, rdi/di, rdx/dx, rip/ip, rsi/si, rsp/sp; 32-bit registers: eax, ebp, ebx, ecx, edx, edi, 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 systemd 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 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**

**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

\texttt{symline:string(addr:long)}

Arguments

\textit{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

\texttt{symname:string(addr:long)}

Arguments

\textit{addr} \hspace{1cm} 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 \textit{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

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

Description

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

**function::task_cpu**

function::task_cpu — The scheduled cpu of the task

**Synopsis**

\[
\text{task_cpu:long}(\text{task:long})
\]

**Arguments**

- \( \text{task} \)  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_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_execname**

function::task_execname — The name of the task

**Synopsis**

```c
    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

\texttt{task_gid:long(task:long)}

Arguments

\texttt{task} \hspace{1em} 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

\[
\text{task\_max\_file\_handles:long(task:long)}
\]

Arguments

\[\text{task} \quad \text{task\_struct\ pointer}\]

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**

\[
\text{task_ns_egid:long}(\text{task:long})
\]

**Arguments**

\[\text{task}\quad \text{task\_struct\ pointeren}\]

**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

\textit{task} \quad \text{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 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

\hspace{0.5cm} task \hspace{0.5cm} task\_struct \hspace{0.5cm} 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

\texttt{task_pid:long(task:long)}

Arguments

\texttt{task} \hspace{1em} task\_struct\hspace{1em} 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

\[
\text{task_tid}:\text{long}(\text{task}:\text{long})
\]

Arguments

\[\text{task} \quad \text{task_struct pointer}\]

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

\[ u_{\text{register}}:\text{long}(\text{name}:\text{string}) \]

Arguments

\[ \text{name} \quad \text{Name of the register to return} \]

Description

Same as \text{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**

```plaintext
text = ubacktrace()
```

**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 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**

```c
uint_arg:int(n:int)
```

**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
class 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

\texttt{usymline:string(addr:long)}

Arguments

\textit{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

\[ \text{cpu\_clock\_ms:long}(\text{cpu:long}) \]

Arguments

\textit{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 \textit{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 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

\texttt{start\_stopwatch(name:string)}

Arguments

- \texttt{name} the stopwatch name

Description

Start stopwatch \texttt{name}. Creates stopwatch \texttt{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

ctime:string(epochsecs:long)

Arguments

ePOCHsecs Number of seconds since epoch (as returned by gettimeofday_s)

Description

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


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”.


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**

```c
 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.
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

    proc_mem_data:long()

Arguments

    None

Description

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

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

Synopsis

proc_mem_data_pid:long(pid:long)

Arguments

pid  The pid of process to examine

Description

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

function::proc_mem_rss — Program resident set size in pages

Synopsis

\[
\text{proc_mem_rss:long()}
\]

Arguments

None

Description

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

function::proc_mem_rss_pid — Program resident set size in pages

Synopsis

proc_mem_rss_pid:long(pid:long)

Arguments

pid The pid of process to examine

Description

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

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

Synopsis

    proc_mem_shr::long()

Arguments

    None

Description

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

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

Synopsis

proc_mem_shr_pid::long(pid::long)

Arguments

pid The pid of process to examine

Description

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

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

Synopsis

proc_mem_size:long()

Arguments

None

Description

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

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

Synopsis

proc_mem_size_pid:long(pid:long)

Arguments

pid The pid of process to examine

Description

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

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

Synopsis

    proc_mem_string::string()

Arguments

    None

Description

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

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

Synopsis

proc_mem_string_pid:string(pid:long)

Arguments

pid The pid of process to examine

Description

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

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

Synopsis

\[
\text{proc\_mem\_txt}: \text{long()}
\]

Arguments

None

Description

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

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

Synopsis

    proc_mem_txt_pid:long(pid:long)

Arguments

    pid   The pid of process to examine

Description

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

value the fault_type returned by vm.page_fault.return

test 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

<table>
<thead>
<tr>
<th>name</th>
<th>name of the probe point</th>
</tr>
</thead>
<tbody>
<tr>
<td>address</td>
<td>the requested address</td>
</tr>
<tr>
<td>length</td>
<td>the length of the memory segment</td>
</tr>
</tbody>
</table>

Context

The process calling brk.
probe::vm.kfree

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

Synopsis

vm.kfree

Values

<table>
<thead>
<tr>
<th>name</th>
<th>name of the probe point</th>
</tr>
</thead>
<tbody>
<tr>
<td>call_site</td>
<td>address of the function calling this kmemory function</td>
</tr>
<tr>
<td>caller_function</td>
<td>name of the caller function.</td>
</tr>
<tr>
<td>ptr</td>
<td>pointer to the kmemory allocated which is returned by kmalloc</td>
</tr>
</tbody>
</table>
probe::vm.kmalloc

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

Synopsis

vm.kmalloc

Values

- *gfp_flags*  type of kmemory to allocate
- *bytes_alloc*  allocated Bytes
- *bytes_req*  requested Bytes
- *ptr*  pointer to the kmemory allocated
- *gfp_flag_name*  type of kmemory to allocate (in String format)
- *name*  name of the probe point
- *call_site*  address of the kmemory function
- *caller_function*  name of the caller function
probe::vm.kmalloc_node

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

Synopsis

vm.kmalloc_node

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ptr</td>
<td>pointer to the kmemory allocated</td>
</tr>
<tr>
<td>bytes_req</td>
<td>requested Bytes</td>
</tr>
<tr>
<td>bytes_alloc</td>
<td>allocated Bytes</td>
</tr>
<tr>
<td>gfp_flags</td>
<td>type of kmemory to allocate</td>
</tr>
<tr>
<td>caller_function</td>
<td>name of the caller function</td>
</tr>
<tr>
<td>call_site</td>
<td>address of the function calling this kmemory function</td>
</tr>
<tr>
<td>name</td>
<td>name of the probe point</td>
</tr>
<tr>
<td>gfp_flag_name</td>
<td>type of kmemory to allocate(in string format)</td>
</tr>
</tbody>
</table>
probe::vm.kmem_cache_alloc

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

Synopsis

vm.kmem_cache_alloc

Values

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

- name: name of the probe point
- gfp_flag_name: type of kmemory to allocate (in string format)
- call_site: address of the function calling this kmemory function
- caller_function: name of the caller function
- bytes_alloc: allocated Bytes
- gfp_flags: type of kmemory to allocate
- ptr: pointer to the kmemory allocated
- bytes_req: requested Bytes
probe::vm.kmem_cache_free

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

Synopsis

vm.kmem_cache_free

Values

<table>
<thead>
<tr>
<th>name</th>
<th>Name of the probe point</th>
</tr>
</thead>
<tbody>
<tr>
<td>caller_function</td>
<td>Name of the caller function.</td>
</tr>
<tr>
<td>call_site</td>
<td>Address of the function calling this kmemory function</td>
</tr>
<tr>
<td>ptr</td>
<td>Pointer to the kmemory allocated which is returned by kmem_cache</td>
</tr>
</tbody>
</table>
probe::vm.mmap — Fires when an mmap is requested

Synopsis

vm.mmap

Values

- **length**: the length of the memory segment
- **name**: name of the probe point
- **address**: the requested address

Context

The process calling mmap.
probe::vm.munmap

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

Synopsis

vm.munmap

Values

- length the length of the memory segment
- name name of the probe point
- address the requested address

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

- **name**: name of the probe point
- **task**: the task being killed

Context

The process that tried to consume excessive memory, and thus triggered the OOM.
probe::vm.pagefault

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

Synopsis

vm.pagefault

Values

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

name name of the probe point

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

Context

The process which triggered the fault
probe::vm.pagefault.return

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

Synopsis

vm.pagefault.return

Values

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

name
name of the probe point
probe::vm.write_shared

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

Synopsis

vm.write_shared

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>address</td>
<td>the address of the shared write</td>
</tr>
<tr>
<td>name</td>
<td>name of the probe point</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>zero</td>
<td>boolean indicating whether it is a zero page (can do a clear instead of a copy)</td>
</tr>
<tr>
<td>name</td>
<td>Name of the probe point</td>
</tr>
<tr>
<td>address</td>
<td>The address of the shared write</td>
</tr>
</tbody>
</table>

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 ZZZZZZZZZ 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 current task

Synopsis

    task_stime:long()

Arguments

    None

Description

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

function::task_stime_tid — System time of the given task

Synopsis

    task_stime_tid:long(tid:long)

Arguments

    tid  Thread id of the given task

Description

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

function::task_utime — User time of the current task

Synopsis

task_utime:long()

Arguments

None

Description

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

    function::task_utime_tid — User time of the given task

Synopsis

    task_utime_tid:long(tid:long)

Arguments

    tid  Thread id of the given task

Description

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

idle  boolean indicating whether current is the idle process

task_next  the process replacing current

task_prev  the process leaving the cpu (same as current)

name  name of the probe point

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

idle

- boolean indicating whether current is the idle process

name

name of the probe point

task_prev

the process that was previously running on this cpu

Context

The resuming process.
### probe::scheduler.ctxswitch

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

#### Synopsis

```
scheduler.ctxswitch
```

#### Values

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prevsk_state</td>
<td>the state of the process to be switched out</td>
</tr>
<tr>
<td>prev_pid</td>
<td>The PID of the process to be switched out</td>
</tr>
<tr>
<td>prev_priority</td>
<td>The priority of the process to be switched out</td>
</tr>
<tr>
<td>nextsk_state</td>
<td>the state of the process to be switched in</td>
</tr>
<tr>
<td>next_pid</td>
<td>The PID of the process to be switched in</td>
</tr>
<tr>
<td>next_priority</td>
<td>The priority of the process to be switched in</td>
</tr>
<tr>
<td>name</td>
<td>name of the probe point</td>
</tr>
<tr>
<td>next_task_name</td>
<td>The name of the process to be switched in</td>
</tr>
<tr>
<td>next_tid</td>
<td>The TID of the process to be switched in</td>
</tr>
<tr>
<td>prev_tid</td>
<td>The TID of the process to be switched out</td>
</tr>
<tr>
<td>prev_task_name</td>
<td>The name of the process to be switched out</td>
</tr>
</tbody>
</table>
probe::scheduler.kthread_stop

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

Synopsis

scheduler.kthread_stop

Values

<table>
<thead>
<tr>
<th>thread_pid</th>
<th>PID of the thread being stopped</th>
</tr>
</thead>
<tbody>
<tr>
<td>thread_priority</td>
<td>priority of the thread</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>name</th>
<th>name of the probe point</th>
</tr>
</thead>
<tbody>
<tr>
<td>return_value</td>
<td>return value after stopping the thread</td>
</tr>
</tbody>
</table>
## probe::scheduler.migrate

probe::scheduler.migrate — Task migrating across cpus

### Synopsis

`scheduler.migrate`

### Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpu_to</td>
<td>the destination cpu</td>
</tr>
<tr>
<td>task</td>
<td>the process that is being migrated</td>
</tr>
<tr>
<td>name</td>
<td>name of the probe point</td>
</tr>
<tr>
<td>pid</td>
<td>PID of the task being migrated</td>
</tr>
<tr>
<td>cpu_from</td>
<td>the original cpu</td>
</tr>
<tr>
<td>priority</td>
<td>priority of the task being migrated</td>
</tr>
</tbody>
</table>
probe::scheduler.process_exit

probe::scheduler.process_exit — Process exiting

Synopsis

scheduler.process_exit

Values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pid</td>
<td>PID of the process exiting</td>
</tr>
<tr>
<td>priority</td>
<td>priority of the process exiting</td>
</tr>
<tr>
<td>name</td>
<td>name of the probe point</td>
</tr>
</tbody>
</table>
probe::scheduler.process_fork

probe::scheduler.process_fork — Process forked

Synopsis

scheduler.process_fork

Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name of the probe point</td>
</tr>
<tr>
<td>parent_pid</td>
<td>PID of the parent process</td>
</tr>
<tr>
<td>child_pid</td>
<td>PID of the child process</td>
</tr>
</tbody>
</table>
probe::scheduler.process_free

Scheduler freeing a data structure for a process

Synopsis

scheduler.process_free

Values

<table>
<thead>
<tr>
<th>name</th>
<th>name of the probe point</th>
</tr>
</thead>
<tbody>
<tr>
<td>pid</td>
<td>PID of the process getting freed</td>
</tr>
<tr>
<td>priority</td>
<td>priority of the process getting freed</td>
</tr>
</tbody>
</table>
probe::scheduler.process_wait

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

Synopsis

scheduler.process_wait

Values

name   name of the probe point
pid    PID of the process scheduler is waiting on
### probe::scheduler.signal_send

probe::scheduler.signal_send — Sending a signal

#### Synopsis

scheduler.signal_send

#### Values

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pid</td>
<td>pid of the process sending signal</td>
</tr>
<tr>
<td>signal_number</td>
<td>signal number</td>
</tr>
<tr>
<td>name</td>
<td>name of the probe point</td>
</tr>
</tbody>
</table>
probe::scheduler.tick

probe::scheduler.tick — Schedulers internal tick, a processes timeslice accounting is updated

Synopsis

scheduler.tick

Values

- **name**  name of the probe point
- **idle**  boolean indicating whether current is the idle process

Context

The process whose accounting will be updated.
probe::scheduler.wait_task

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

Synopsis

scheduler.wait_task

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_pid</td>
<td>PID of the task the scheduler is waiting on</td>
</tr>
<tr>
<td>name</td>
<td>name of the probe point</td>
</tr>
<tr>
<td>task_priority</td>
<td>priority of the task</td>
</tr>
</tbody>
</table>
probe::scheduler.wakeup

probe::scheduler.wakeup — Task is woken up

Synopsis

scheduler.wakeup

Values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_state</td>
<td>state of the task being woken up</td>
</tr>
<tr>
<td>task_priority</td>
<td>priority of the task being woken up</td>
</tr>
<tr>
<td>task_cpu</td>
<td>cpu of the task being woken up</td>
</tr>
<tr>
<td>task_tid</td>
<td>tid of the task being woken up</td>
</tr>
<tr>
<td>name</td>
<td>name of the probe point</td>
</tr>
<tr>
<td>task_pid</td>
<td>PID of the task being woken up</td>
</tr>
</tbody>
</table>
probe::scheduler.wakeup_new

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

Synopsis

scheduler.wakeup_new

Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_pid</td>
<td>PID of the new task woken up</td>
</tr>
<tr>
<td>name</td>
<td>name of the probe point</td>
</tr>
<tr>
<td>task_tid</td>
<td>TID of the new task woken up</td>
</tr>
<tr>
<td>task_cpu</td>
<td>cpu of the task woken up</td>
</tr>
<tr>
<td>task_priority</td>
<td>priority of the new task</td>
</tr>
<tr>
<td>task_state</td>
<td>state of the task woken up</td>
</tr>
</tbody>
</table>
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**

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

**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

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

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

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

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**

```c
ioblock_trace.end
```

**Values**

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

**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**

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

**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**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>disk_major</td>
<td>Disk major no of request.</td>
</tr>
<tr>
<td>disk_minor</td>
<td>Disk minor number of request.</td>
</tr>
<tr>
<td>q</td>
<td>Pointer to request queue.</td>
</tr>
<tr>
<td>rq</td>
<td>Address of request.</td>
</tr>
<tr>
<td>rq_flags</td>
<td>Request flags.</td>
</tr>
<tr>
<td>elevator_name</td>
<td>The type of I/O elevator currently enabled.</td>
</tr>
</tbody>
</table>
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

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

disk_minor
Disk minor number of request.

disk_major
Disk major no of request.

elevator_name
The type of I/O elevator currently enabled.

rq_flags
Request flags.

rq
Address of request.

name
Name of the probe point

q
Pointer to request queue.
probe::ioscheduler.elv_completed_request

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

Synopsis

ioscheduler.elv_completed_request

Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the probe point</td>
</tr>
<tr>
<td>rq_flags</td>
<td>Request flags</td>
</tr>
<tr>
<td>rq</td>
<td>Address of the request</td>
</tr>
<tr>
<td>elevator_name</td>
<td>The type of I/O elevator currently enabled</td>
</tr>
<tr>
<td>disk_major</td>
<td>Disk major number of the request</td>
</tr>
<tr>
<td>disk_minor</td>
<td>Disk minor number of the request</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elevator_name</td>
<td>The type of I/O elevator currently enabled</td>
</tr>
<tr>
<td>name</td>
<td>Name of the probe point</td>
</tr>
</tbody>
</table>
**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**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the probe point</td>
</tr>
<tr>
<td>rq_flags</td>
<td>Request flags</td>
</tr>
<tr>
<td>rq</td>
<td>Address of the request</td>
</tr>
<tr>
<td>disk_major</td>
<td>Disk major number of the request</td>
</tr>
<tr>
<td>disk_minor</td>
<td>Disk minor number of the request</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the probe point</td>
</tr>
<tr>
<td>elevator_name</td>
<td>The type of I/O elevator currently enabled.</td>
</tr>
<tr>
<td>rq_flags</td>
<td>Request flags.</td>
</tr>
<tr>
<td>rq</td>
<td>Address of request.</td>
</tr>
<tr>
<td>disk_major</td>
<td>Disk major no of request.</td>
</tr>
<tr>
<td>disk_minor</td>
<td>Disk minor number of request.</td>
</tr>
</tbody>
</table>
**probe::ioscheduler_trace.elv_completed_request**

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

**Synopsis**

ioscheduler_trace.elv_completed_request

**Values**

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

**Description**

completed.
probe::ioscheduler_trace.elv_issue_request

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

Synopsis

ioscheduler_trace.elv_issue_request

Values

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

Description

scheduled.
probe::ioscheduler_trace.elv_requeue_request

Synopsis

probe::ioscheduler_trace.elv_requeue_request — Fires when a request is put back on the queue, when the hardware cannot accept more requests.

Values

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

Description

put back on the queue, when the hardware cannot accept more requests.
probe::ioscheduler_trace.plug

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

Synopsis

ioscheduler_trace.plug

Values

rq_queue request queue
name 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

<table>
<thead>
<tr>
<th>name</th>
<th>Name of the probe point</th>
</tr>
</thead>
<tbody>
<tr>
<td>rq_queue</td>
<td>request queue</td>
</tr>
</tbody>
</table>

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**

- `name`  Name of the probe point
- `rq_queue`  request queue

**Description**

with a request queue expires.
Chapter 10. SCSI Tapset

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

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

Synopsis

scsi.iocompleted

Values

data_direction

data_direction specifies whether this command is from/to the device

goodbytes

The bytes completed

data_direction_str

Data direction, as a string

device_state_str

The current state of the device, as a string

host_no

The host number

channel

The channel number

req_addr

The current struct request pointer, as a number

device_state

The current state of the device

lun

The lun number

dev_id

The scsi device id
probescisi.iiodispatching

probescisi.iiodispatching — SCSI mid-layer dispatched low-level SCSI command

Synopsis

scsi.iiodispatching

Values

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

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

Synopsis

scsi.iodone

Values

channel

The channel number

req_addr

The current struct request pointer, as a number

data_direction

The data_direction specifies whether this command is from/to the device.

data_direction_str

Data direction, as a string

device_state_str

The current state of the device, as a string

host_no

The host number

lun

The lun number

dev_id

The scsi device id

scsi_timer_pending

1 if a timer is pending on this request

device_state

The current state of the device
probe::scsi.ioentry

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

Synopsis

scsi.ioentry

Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>device_state_str</td>
<td>The current state of the device, as a string</td>
</tr>
<tr>
<td>device_state</td>
<td>The current state of the device</td>
</tr>
<tr>
<td>disk_major</td>
<td>The major number of the disk (-1 if no information)</td>
</tr>
<tr>
<td>disk_minor</td>
<td>The minor number of the disk (-1 if no information)</td>
</tr>
<tr>
<td>req_addr</td>
<td>The current struct request pointer, as a number</td>
</tr>
</tbody>
</table>
probe::scsi.ioexecute

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

Synopsis

scsi.ioexecute

Values

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

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

Synopsis

  scsi.set_state

Values

  channel              The channel number
  state_str            The new state of the device, as a string
  old_state            The current state of the device
  state                The new state of the device
  host_no              The host number
  lun                  The lun number
  dev_id               The scsi device id
  old_state_str        The current state of the device, as a string
Chapter 11. TTY Tapset

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

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

Synopsis

tty.init

Values

module the module name

driver_name the driver name

name the driver .dev_name name
**probe::tty.ioctl**

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

**Synopsis**

tty.ioctl

**Values**

- *cmd*  the ioctl command
- *name*  the file name
- *arg*  the ioctl argument
probe::tty.open

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

Synopsis

tty.open

Values

file_mode       the file mode
inode_state    the inode state
file_name      the file name
inode_number   the inode number
file_flags     the file flags
inode_flags    the inode flags
probe::tty.poll

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

Synopsis

tty.poll

Values

wait_key the wait queue key
file_name the tty file name
probe::tty.read

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

Synopsis

tty.read

Values

buffer the buffer that will receive the characters
file_name the file name lreated to the tty
driver_name the driver name
nr The amount of characters to be read
probes::tty.receive

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

Synopsis

tty.receive

Values

<table>
<thead>
<tr>
<th>id</th>
<th>the tty id</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>The tty Index</td>
</tr>
<tr>
<td>cp</td>
<td>the buffer that was received</td>
</tr>
<tr>
<td>count</td>
<td>The amount of characters received</td>
</tr>
<tr>
<td>fp</td>
<td>The flag buffer</td>
</tr>
<tr>
<td>name</td>
<td>the name of the module file</td>
</tr>
<tr>
<td>driver_name</td>
<td>the driver name</td>
</tr>
</tbody>
</table>
**probe::tty.register**

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

**Synopsis**

```
tty.register
```

**Values**

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>index</code></td>
<td>the tty index requested</td>
</tr>
<tr>
<td><code>name</code></td>
<td>the driver .dev_name name</td>
</tr>
<tr>
<td><code>module</code></td>
<td>the module name</td>
</tr>
<tr>
<td><code>driver_name</code></td>
<td>the driver name</td>
</tr>
</tbody>
</table>
probe::tty.release

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

Synopsis

tty.release

Values

inode_state    the inode state
file_name      the file name
inode_number   the inode number
file_flags     the file flags
inode_flags    the inode flags
file_mode      the file mode
probe::tty.resize

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

Synopsis

tty.resize

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>old_ypixel</td>
<td>the old ypixel</td>
</tr>
<tr>
<td>new_row</td>
<td>the new row value</td>
</tr>
<tr>
<td>old_col</td>
<td>the old col value</td>
</tr>
<tr>
<td>name</td>
<td>the tty name</td>
</tr>
<tr>
<td>new_xpixel</td>
<td>the new xpixel value</td>
</tr>
<tr>
<td>new_ypixel</td>
<td>the new ypixel value</td>
</tr>
<tr>
<td>old_xpixel</td>
<td>the old xpixel</td>
</tr>
<tr>
<td>old_row</td>
<td>the old row value</td>
</tr>
<tr>
<td>new_col</td>
<td>the new col value</td>
</tr>
</tbody>
</table>
probe::tty.unregister

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

Synopsis

tty.unregister

Values

driver_name the driver name
module the module name
name the driver .dev_name name
index the tty index requested
probe::tty.write

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

Synopsis

tty.write

Values

<table>
<thead>
<tr>
<th>driver_name</th>
<th>the driver name</th>
</tr>
</thead>
<tbody>
<tr>
<td>nr</td>
<td>The amount of characters</td>
</tr>
<tr>
<td>buffer</td>
<td>the buffer that will be written</td>
</tr>
<tr>
<td>file_name</td>
<td>the file name lreated to the tty</td>
</tr>
</tbody>
</table>
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**

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

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

Synopsis

irq_handler.exit

Values

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

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

Synopsis

softirq.entry

Values

\( h \)  \quad \text{struct softirq\_action* for current pending softirq}

\( vec \)  \quad \text{softirq\_action vector}

\( vec\_nr \)  \quad \text{softirq vector number}

\( action \)  \quad \text{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

h struct softirq_action* for just executed softirq
vec_nr softirq vector number
vec softirq_action vector
action pointer to softirq handler that just finished execution
probe::workqueue.create

probe::workqueue.create — Creating a new workqueue

Synopsis

workqueue.create

Values

\textit{wq\_thread} \hspace{1em} \text{task\_struct of the workqueue thread}

\textit{cpu} \hspace{1em} \text{cpu for which the worker thread is created}
probe::workqueue.destroy

probe::workqueue.destroy — Destroying workqueue

Synopsis

workqueue.destroy

Values

\texttt{wq\_thread} \hspace{1em} \texttt{task\_struct\ of\ the\ workqueue\ thread}
**probe::workqueue.execute**

probe::workqueue.execute — Executing deferred work

**Synopsis**

`workqueue.execute`

**Values**

- `wq_thread` task_struct of the workqueue thread
- `work` work_struct* being executed
- `work_func` pointer to handler function
**probe::workqueue.insert**

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

**Synopsis**

workqueue.insert

**Values**

- **work_func** — pointer to handler function
- **work** — work_struct* being queued
- **wq_thread** — task_struct of the workqueue thread
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

addr     the IP address
family   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

\[ \text{ntohl}: \text{long}(x:\text{long}) \]

Arguments

\[ x \text{ Value to convert} \]
function::ntohll

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

Synopsis

ntohll::long(x:long)

Arguments

x Value to convert
**function::ntohs**

_convert 16-bit short from network to host order_

**Synopsis**

\[ \text{ntohs}: \text{long}(x:\text{long}) \]

**Arguments**

\[ x \quad \text{Value to convert} \]
probe::netdev.change_mac

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

Synopsis

netdev.change_mac

Values

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mac_len</td>
<td>The MAC length</td>
</tr>
<tr>
<td>dev_name</td>
<td>The device that will have the MAC changed</td>
</tr>
<tr>
<td>new_mac</td>
<td>The new MAC address</td>
</tr>
<tr>
<td>old_mac</td>
<td>The current MAC address</td>
</tr>
</tbody>
</table>
**probe::netdev.change_mtu**

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

## Synopsis

```c
netdev.change_mtu
```

## Values

- `new_mtu`    The new MTU
- `old_mtu`    The current MTU
- `dev_name`   The device that will have the MTU changed
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

flags The new flags
dev_name The device that will be changed
probe::netdev.close

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

Synopsis

netdev.close

Values

\textit{dev\_name} \hspace{1cm} 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

\texttt{dev\_name} \quad \text{The device that is going to provide the statistics}
probe::netdev.hard_transmit

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

Synopsis

netdev.hard_transmit

Values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>The length of the transmit buffer.</td>
</tr>
<tr>
<td>truesize</td>
<td>The size of the data to be transmitted.</td>
</tr>
<tr>
<td>dev_name</td>
<td>The device scheduled to transmit.</td>
</tr>
<tr>
<td>protocol</td>
<td>The protocol used in the transmission.</td>
</tr>
</tbody>
</table>
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

dev_name  The device that is going to be opened
probe::netdev.receive

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

Synopsis

netdev.receive

Values

protocol Protocol of received packet.
dev_name The name of the device. e.g: eth0, ath1.
length The length of the receiving buffer.
probe::netdev.register

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

Synopsis

netdev.register

Values

\texttt{dev\_name} \quad \text{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

<table>
<thead>
<tr>
<th>protocol</th>
<th>The packet protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>dev_name</td>
<td>The device received the packet</td>
</tr>
</tbody>
</table>
probe::netdev.set_promiscuity

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

Synopsis

netdev.set_promiscuity

Values

`dev_name` The device that is entering/leaving promiscuity mode
`disable` If the device is leaving promiscuity mode
`inc` Count the number of promiscuity openers
`enable` If the device is entering promiscuity mode
 Networking Tapset

probe::netdev.transmit

probe::netdev.transmit — Network device transmitting buffer

Synopsis

netdev.transmit

Values

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

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

Synopsis

netdev.unregister

Values

dev_name The device that is going to be unregistered
probes::netfilter.arp.forward

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

Synopsis

netfilter.arp.forward

Values

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

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

Synopsis

genfilter.arp.in

Values

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

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

## Synopsis

netfilter.arp.out

## Values

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

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

Synopsis

netfilter.bridge.forward

Values

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nf_stolen</td>
<td>Constant used to signify a 'stolen' verdict</td>
</tr>
<tr>
<td>llcproto_stp</td>
<td>Constant used to signify Bridge Spanning Tree Protocol packet</td>
</tr>
<tr>
<td>br_mac</td>
<td>Bridge MAC address</td>
</tr>
<tr>
<td>protocol</td>
<td>Packet protocol</td>
</tr>
<tr>
<td>br_rmac</td>
<td>Root bridge MAC address</td>
</tr>
<tr>
<td>br_poid</td>
<td>Port identifier</td>
</tr>
<tr>
<td>br_fd</td>
<td>Forward delay in 1/256 secs</td>
</tr>
<tr>
<td>br_type</td>
<td>BPDU type</td>
</tr>
<tr>
<td>nf_repeat</td>
<td>Constant used to signify a 'repeat' verdict</td>
</tr>
<tr>
<td>br_flags</td>
<td>BPDU flags</td>
</tr>
<tr>
<td>br_cost</td>
<td>Total cost from transmitting bridge to root</td>
</tr>
<tr>
<td>brhdr</td>
<td>Address of bridge header</td>
</tr>
<tr>
<td>outdev</td>
<td>Address of net_device representing output device, 0 if unknown</td>
</tr>
<tr>
<td>indev</td>
<td>Address of net_device representing input device, 0 if unknown</td>
</tr>
<tr>
<td>br_hptime</td>
<td>Hello time in 1/256 secs</td>
</tr>
<tr>
<td>br_max</td>
<td>Max age in 1/256 secs</td>
</tr>
<tr>
<td>llcpdu</td>
<td>Address of LLC Protocol Data Unit</td>
</tr>
<tr>
<td>length</td>
<td>The length of the packet buffer contents, in bytes</td>
</tr>
<tr>
<td>br_prid</td>
<td>Protocol identifier</td>
</tr>
<tr>
<td>nf_queue</td>
<td>Constant used to signify a 'queue' verdict</td>
</tr>
<tr>
<td>br_rid</td>
<td>Identity of root bridge</td>
</tr>
<tr>
<td>nf_stop</td>
<td>Constant used to signify a 'stop' verdict</td>
</tr>
<tr>
<td>br_bid</td>
<td>Identity of bridge</td>
</tr>
<tr>
<td>pf</td>
<td>Protocol family — always “bridge”</td>
</tr>
</tbody>
</table>
### Networking Tapset

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nf_accept</code></td>
<td>Constant used to signify an 'accept' verdict</td>
</tr>
<tr>
<td><code>outdev_name</code></td>
<td>Name of network device packet will be routed to (if known)</td>
</tr>
<tr>
<td><code>br_vid</code></td>
<td>Protocol version identifier</td>
</tr>
<tr>
<td><code>indev_name</code></td>
<td>Name of network device packet was received on (if known)</td>
</tr>
<tr>
<td><code>nf_drop</code></td>
<td>Constant used to signify a 'drop' verdict</td>
</tr>
<tr>
<td><code>br_msg</code></td>
<td>Message age in 1/256 secs</td>
</tr>
</tbody>
</table>
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

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

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

probe::netfilter.bridge.post_routing

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

Synopsis

netfilter.bridge.post_routing

Values

<table>
<thead>
<tr>
<th>indev_name</th>
<th>Name of network device packet was received on (if known)</th>
</tr>
</thead>
<tbody>
<tr>
<td>br_vid</td>
<td>Protocol version identifier</td>
</tr>
<tr>
<td>nf_drop</td>
<td>Constant used to signify a 'drop' verdict</td>
</tr>
<tr>
<td>br_msg</td>
<td>Message age in 1/256 secs</td>
</tr>
<tr>
<td>outdev_name</td>
<td>Name of network device packet will be routed to (if known)</td>
</tr>
<tr>
<td>br_rid</td>
<td>Identity of root bridge</td>
</tr>
<tr>
<td>nf_queue</td>
<td>Constant used to signify a 'queue' verdict</td>
</tr>
<tr>
<td>br_prid</td>
<td>Protocol identifier</td>
</tr>
<tr>
<td>nf_stop</td>
<td>Constant used to signify a 'stop' verdict</td>
</tr>
<tr>
<td>br_bid</td>
<td>Identity of bridge</td>
</tr>
<tr>
<td>pf</td>
<td>Protocol family -- always “bridge”</td>
</tr>
<tr>
<td>nf_accept</td>
<td>Constant used to signify an 'accept' verdict</td>
</tr>
<tr>
<td>br_cost</td>
<td>Total cost from transmitting bridge to root</td>
</tr>
<tr>
<td>brhdr</td>
<td>Address of bridge header</td>
</tr>
<tr>
<td>outdev</td>
<td>Address of net_device representing output device, 0 if unknown</td>
</tr>
<tr>
<td>indev</td>
<td>Address of net_device representing input device, 0 if unknown</td>
</tr>
<tr>
<td>br_hitime</td>
<td>Hello time in 1/256 secs</td>
</tr>
<tr>
<td>br_max</td>
<td>Max age in 1/256 secs</td>
</tr>
<tr>
<td>length</td>
<td>The length of the packet buffer contents, in bytes</td>
</tr>
<tr>
<td>llcpdu</td>
<td>Address of LLC Protocol Data Unit</td>
</tr>
<tr>
<td>br_type</td>
<td>BPDU type</td>
</tr>
<tr>
<td>nf_repeat</td>
<td>Constant used to signify a 'repeat' verdict</td>
</tr>
<tr>
<td>br_flags</td>
<td>BPDU flags</td>
</tr>
<tr>
<td>br_fd</td>
<td>Forward delay in 1/256 secs</td>
</tr>
</tbody>
</table>

299
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>br_mac</code></td>
<td>Bridge MAC address</td>
</tr>
<tr>
<td><code>protocol</code></td>
<td>Packet protocol</td>
</tr>
<tr>
<td><code>br_rmac</code></td>
<td>Root bridge MAC address</td>
</tr>
<tr>
<td><code>br_poid</code></td>
<td>Port identifier</td>
</tr>
<tr>
<td><code>nf_stolen</code></td>
<td>Constant used to signify a 'stolen' verdict</td>
</tr>
<tr>
<td><code>llcproto_stp</code></td>
<td>Constant used to signify Bridge Spanning Tree Protocol packet</td>
</tr>
</tbody>
</table>
probe::netfilter.bridge.pre_routing

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

Synopsis

netfilter.bridge.pre_routing

Values

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

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

Synopsis

netfilter.ip.forward

Values

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

<table>
<thead>
<tr>
<th>indev_name</th>
<th>Name of network device packet was received on (if known)</th>
</tr>
</thead>
<tbody>
<tr>
<td>nf_drop</td>
<td>Constant used to signify a 'drop' verdict</td>
</tr>
<tr>
<td>iphdr</td>
<td>Address of IP header</td>
</tr>
</tbody>
</table>
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

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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nf_queue</code></td>
<td>Constant used to signify a 'queue' verdict</td>
</tr>
<tr>
<td><code>nf_accept</code></td>
<td>Constant used to signify an 'accept' verdict</td>
</tr>
<tr>
<td><code>pf</code></td>
<td>Protocol family -- either “ipv4” or “ipv6”</td>
</tr>
</tbody>
</table>
Networking Tapset

probe::netfilter.ip.local_out

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

Synopsis

netfilter.ip.local_out

Values

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

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

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

**Synopsis**

netfilter.ip.pre_routing

**Values**

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

nf_repeat  Constant used to signify a 'repeat' verdict
rst       TCP RST flag (if protocol is TCP; ipv4 only)
daddr     A string representing the destination IP address
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

- **old_vers**: the version of old RPC program
- **servername**: the server machine name
- **old_prog**: the number of old RPC program
- **prog**: the number of new RPC program
- **old_progname**: the name of old RPC program
- **vers**: the version of new RPC program
- **progname**: the name 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

dead        whether this client is abandoned
servername  the server machine name
xid         current transmission id
flags       flags
prot        the IP protocol number
proc        the procedure number in this RPC call
procname    the procedure name in this RPC call
port        the port number
vers        the RPC program version number
progsname   the RPC program name
prog        the RPC program number
**probe::sunrpc.clnt.call_sync**

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

**Synopsis**

sunrpc.clnt.call_sync

**Values**

- **prog**  
  the RPC program number
- **progname**  
  the RPC program name
- **port**  
  the port number
- **vers**  
  the RPC program version number
- **procname**  
  the procedure name in this RPC call
- **prot**  
  the IP protocol number
- **proc**  
  the procedure number in this RPC call
- **flags**  
  flags
- **xid**  
  current transmission id
- **dead**  
  whether this client is abandoned
- **servername**  
  the server machine name
probe::sunrpc.clnt.clone_client

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

Synopsis

sunrpc.clnt.clone_client

Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prot</td>
<td>the IP protocol number</td>
</tr>
<tr>
<td>authflavor</td>
<td>the authentication flavor</td>
</tr>
<tr>
<td>progname</td>
<td>the RPC program name</td>
</tr>
<tr>
<td>prog</td>
<td>the RPC program number</td>
</tr>
<tr>
<td>servername</td>
<td>the server machine name</td>
</tr>
<tr>
<td>port</td>
<td>the port number</td>
</tr>
<tr>
<td>vers</td>
<td>the RPC program version number</td>
</tr>
</tbody>
</table>
probe::sunrpc.clnt.create_client

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

Synopsis

sunrpc.clnt.create_client

Values

prot the IP protocol number
programe the RPC program name
authflavor the authentication flavor
port the port number
vers the RPC program version number
prog the RPC program number
servername the server machine name
probe::sunrpc.clnt.restart_call

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

Synopsis

sunrpc.clnt.restart_call

Values

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>servername</td>
<td>the server machine name</td>
</tr>
<tr>
<td>tk_flags</td>
<td>the task flags</td>
</tr>
<tr>
<td>xid</td>
<td>the transmission id</td>
</tr>
<tr>
<td>tk_pid</td>
<td>the debugging aid of task</td>
</tr>
<tr>
<td>prog</td>
<td>the RPC program number</td>
</tr>
<tr>
<td>tk_priority</td>
<td>the task priority</td>
</tr>
<tr>
<td>tk_runstate</td>
<td>the task run status</td>
</tr>
</tbody>
</table>
probe::sunrpc.clnt.shutdown_client

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

Synopsis

sunrpc.clnt.shutdown_client

Values

om_execute          the RPC execution jiffies
servername          the server machine name
om_bytes_sent       the count of bytes out
clones              the number of clones
prot                 the IP protocol number
om_ntrans           the count of RPC transmissions
om_ops              the count of operations
authflavor          the authentication flavor
om_bytes_recv       the count of bytes in
port                 the port number
rpcctnt             the count of RPC calls
vers                 the RPC program version number
progrname            the RPC program name
prog                 the RPC program number
tasks                the number of references
om_queue             the jiffies queued for xmit
netreconn            the count of reconnections
om_rtt               the RPC RTT jiffies
probe::sunrpc.sched.delay

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

Synopsis

sunrpc.sched.delay

Values

| prog     | the program number in the RPC call |
| vers     | the program version in the RPC call |
| prot     | the IP protocol in the RPC call |
| tk_flags | the flags of the task |
| xid      | the transmission id in the RPC call |
| delay    | the time delayed |
| tk_pid   | the debugging id of the task |
probe::sunrpc.sched.execute

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

Synopsis

sunrpc.sched.execute

Values

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

\[\begin{array}{ll}
\text{xid} & \text{the transmission id in the RPC call} \\
\text{tk_flags} & \text{the flags of the task} \\
\text{prot} & \text{the IP protocol in the RPC call} \\
\text{vers} & \text{the program version in the RPC call} \\
\text{prog} & \text{the program number in the RPC call}
\end{array}\]
probe::sunrpc.sched.release_task

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

Synopsis

sunrpc.sched.release_task

Values

\text{\texttt{xid}} \quad \text{the transmission id in the RPC call}
\text{\texttt{tk\_flags}} \quad \text{the flags of the task}
\text{\texttt{prot}} \quad \text{the IP protocol in the RPC call}
\text{\texttt{vers}} \quad \text{the program version in the RPC call}
\text{\texttt{prog}} \quad \text{the program number in the RPC call}

Description

\texttt{rpc\_release\_task} function might not be found for a particular kernel. So, if we can't find it, just return '\texttt{-1}' for everything.
probe::sunrpc.svc.create

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

Synopsis

sunrpc.svc.create

Values

\texttt{pg\_nvers} \hspace{1cm} the number of supported versions
\texttt{prog} \hspace{1cm} the number of the program
\texttt{progname} \hspace{1cm} the name of the program
\texttt{bufsize} \hspace{1cm} the buffer size
probe::sunrpc.svc.destroy

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

Synopsis

sunrpc.svc.destroy

Values

netcnt the count of received RPC requests
rpcbadfmt the count of requests dropped for bad formats
nettcpconn the count of accepted TCP connections
rpcbadauth the count of requests dropped for authentication failure
sv_name the service name
sv_prog the number of the program
sv_progname the name of the program
rpccnt the count of valid RPC requests
sv_nrthreads the number of concurrent threads
probe::sunrpc.svc.drop

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

Synopsis

sunrpc.svc.drop

Values

\[ \begin{align*}
\text{rq_prot} & \quad \text{the IP protocol of the request} \\
\text{rq_proc} & \quad \text{the procedure number in the request} \\
\text{rq_vers} & \quad \text{the program version in the request} \\
\text{sv_name} & \quad \text{the service name} \\
\text{peer_ip} & \quad \text{the peer address where the request is from} \\
\text{rq_prog} & \quad \text{the program number in the request} \\
\text{rq_xid} & \quad \text{the transmission id in the request}
\end{align*} \]
probe::sunrpc.svc.process — Process an RPC request

Synopsis

sunrpc.svc.process

Values

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

| sv_prog     | the number of the program |
| sv_nrthreads| the number of concurrent threads |
| sv_name     | the service name |
| timeout     | 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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>the port number</td>
</tr>
<tr>
<td>programe</td>
<td>the name of the program</td>
</tr>
<tr>
<td>sv_name</td>
<td>the service name</td>
</tr>
<tr>
<td>prog</td>
<td>the number of the program</td>
</tr>
<tr>
<td>prot</td>
<td>the IP protocol number</td>
</tr>
</tbody>
</table>

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

rq_prot  the IP protocol of the request
peer_ip  the peer address where the request is from
rq_prog  the program number in the request
rq_xid   the transmission id in the request
rq_proc  the procedure number in the request
rq_vers  the program version in the request
sv_name  the service name
probe::tcp.disconnect — TCP socket disconnection

Synopsis

tcp.disconnect

Values

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

Context

The process which disconnects tcp
probe::tcp.disconnect.return

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

Synopsis

tcp.disconnect.return

Values

- ret   Error code (0: no error)
- name  Name of this probe

Context

The process which disconnects tcp
probe::tcp.receive

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

Synopsis

tcp.receive

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rst</td>
<td>TCP RST flag</td>
</tr>
<tr>
<td>urg</td>
<td>TCP URG flag</td>
</tr>
<tr>
<td>fin</td>
<td>TCP FIN flag</td>
</tr>
<tr>
<td>syn</td>
<td>TCP SYN flag</td>
</tr>
<tr>
<td>ack</td>
<td>TCP ACK flag</td>
</tr>
<tr>
<td>protocol</td>
<td>Packet protocol from driver</td>
</tr>
<tr>
<td>daddr</td>
<td>A string representing the destination IP address</td>
</tr>
<tr>
<td>saddr</td>
<td>A string representing the source IP address</td>
</tr>
<tr>
<td>family</td>
<td>IP address family</td>
</tr>
<tr>
<td>psh</td>
<td>TCP PSH flag</td>
</tr>
<tr>
<td>name</td>
<td>Name of the probe point</td>
</tr>
<tr>
<td>sport</td>
<td>TCP source port</td>
</tr>
<tr>
<td>iphdr</td>
<td>IP header address</td>
</tr>
<tr>
<td>dport</td>
<td>TCP destination port</td>
</tr>
</tbody>
</table>
probe::tcp.recvmsg

probe::tcp.recvmsg — Receiving TCP message

Synopsis
tcp.recvmsg

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>saddr</td>
<td>A string representing the source IP address</td>
</tr>
<tr>
<td>family</td>
<td>IP address family</td>
</tr>
<tr>
<td>sock</td>
<td>Network socket</td>
</tr>
<tr>
<td>sport</td>
<td>TCP source port</td>
</tr>
<tr>
<td>dport</td>
<td>TCP destination port</td>
</tr>
<tr>
<td>name</td>
<td>Name of this probe</td>
</tr>
<tr>
<td>size</td>
<td>Number of bytes to be received</td>
</tr>
<tr>
<td>daddr</td>
<td>A string representing the destination IP address</td>
</tr>
</tbody>
</table>

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

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

Context

The process which receives a tcp message
**probe::tcp.sendmsg**

probe::tcp.sendmsg — Sending a tcp message

**Synopsis**

```c
tcp.sendmsg
```

**Values**

- `name`: Name of this probe
- `family`: IP address family
- `size`: Number of bytes to send
- `sock`: 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**

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

**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

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

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**

- **daddr**: A string representing the destination IP address
- **ret**: Error code (0: no error)
- **family**: IP address family
- **dport**: UDP destination port
- **saddr**: A string representing the source IP address
- **sport**: UDP source port
- **name**: 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

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

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

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

<table>
<thead>
<tr>
<th>name</th>
<th>The name of this probe</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>Number of bytes sent by the process</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>protocol</th>
<th>Protocol value</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>Message size in bytes</td>
</tr>
<tr>
<td>state</td>
<td>Socket state value</td>
</tr>
<tr>
<td>name</td>
<td>Name of this probe</td>
</tr>
<tr>
<td>type</td>
<td>Socket type value</td>
</tr>
<tr>
<td>family</td>
<td>Protocol family value</td>
</tr>
<tr>
<td>flags</td>
<td>Socket flags value</td>
</tr>
</tbody>
</table>

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

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

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>protocol</td>
<td>Protocol value</td>
</tr>
<tr>
<td>state</td>
<td>Socket state value</td>
</tr>
<tr>
<td>size</td>
<td>Message size in bytes</td>
</tr>
<tr>
<td>family</td>
<td>Protocol family value</td>
</tr>
<tr>
<td>type</td>
<td>Socket type value</td>
</tr>
<tr>
<td>name</td>
<td>Name of this probe</td>
</tr>
<tr>
<td>flags</td>
<td>Socket flags value</td>
</tr>
</tbody>
</table>

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

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

Context

The message receiver.

Description

Fires at the conclusion of sending a message on a socket via the sock_aio_write function
probe::socket.close

probe::socket.close — Close a socket

Synopsis

socket.close

Values

<table>
<thead>
<tr>
<th>name</th>
<th>Name of this probe</th>
</tr>
</thead>
<tbody>
<tr>
<td>family</td>
<td>Protocol family value</td>
</tr>
<tr>
<td>type</td>
<td>Socket type value</td>
</tr>
<tr>
<td>flags</td>
<td>Socket flags value</td>
</tr>
<tr>
<td>protocol</td>
<td>Protocol value</td>
</tr>
<tr>
<td>state</td>
<td>Socket state value</td>
</tr>
</tbody>
</table>

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**

- `protocol` Protocol value
- `family` Protocol family value
- `type` Socket type value
- `name` Name of this probe
- `requester` 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

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

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>protocol</td>
<td>Protocol value</td>
</tr>
<tr>
<td>size</td>
<td>Message size in bytes</td>
</tr>
<tr>
<td>state</td>
<td>Socket state value</td>
</tr>
<tr>
<td>name</td>
<td>Name of this probe</td>
</tr>
<tr>
<td>type</td>
<td>Socket type value</td>
</tr>
<tr>
<td>family</td>
<td>Protocol family value</td>
</tr>
<tr>
<td>flags</td>
<td>Socket flags value</td>
</tr>
</tbody>
</table>

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

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

Context

The message receiver.

Description

Fires at the conclusion of receiving a message on a socket via the sock_read_iter function
**probe::socket.readv**

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

**Synopsis**

socket.readv

**Values**

- **state**: Socket state value
- **size**: Message size in bytes
- **protocol**: Protocol value
- **flags**: Socket flags value
- **family**: Protocol family value
- **type**: Socket type value
- **name**: Name of this probe

**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

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

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

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

Context

The message receiver
probe::socket.recvmsg

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

Synopsis

socket.recvmsg

Values

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

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

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

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

Context

The message sender
probe::socket.sendmsg

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

Synopsis

socket.sendmsg

Values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><em>state</em></td>
<td>Socket state value</td>
</tr>
<tr>
<td><em>size</em></td>
<td>Message size in bytes</td>
</tr>
<tr>
<td><em>protocol</em></td>
<td>Protocol value</td>
</tr>
<tr>
<td><em>flags</em></td>
<td>Socket flags value</td>
</tr>
<tr>
<td><em>name</em></td>
<td>Name of this probe</td>
</tr>
<tr>
<td><em>family</em></td>
<td>Protocol family value</td>
</tr>
<tr>
<td><em>type</em></td>
<td>Socket type value</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>name</th>
<th>Name of this probe</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Socket type value</td>
</tr>
<tr>
<td>family</td>
<td>Protocol family value</td>
</tr>
<tr>
<td>flags</td>
<td>Socket flags value</td>
</tr>
<tr>
<td>protocol</td>
<td>Protocol value</td>
</tr>
<tr>
<td>size</td>
<td>Message size in bytes</td>
</tr>
<tr>
<td>state</td>
<td>Socket state value</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>protocol</td>
<td>Protocol value</td>
</tr>
<tr>
<td>size</td>
<td>Message size in bytes</td>
</tr>
<tr>
<td>state</td>
<td>Socket state value</td>
</tr>
<tr>
<td>type</td>
<td>Socket type value</td>
</tr>
<tr>
<td>family</td>
<td>Protocol family value</td>
</tr>
<tr>
<td>name</td>
<td>Name of this probe</td>
</tr>
<tr>
<td>flags</td>
<td>Socket flags value</td>
</tr>
</tbody>
</table>

**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

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

- **skb**: pointer to the struct sk_buff
- **op**: value to be counted if skb passes the filter
- **SourceIsLocal**: 1 is local operation and 0 is non-local operation

Description

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

skb
   pointer to a struct sk_buff

SourceIsLocal
   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

- `skb` : pointer to a struct sk_buff
- `SourceIsLocal` : 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

skb

pointer to a struct sk_buff

SourceIsLocal

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**

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

**Arguments**

- `skb` pointer to a struct `sk_buff`
- `SourceIsLocal` flag to indicate whether local operation

**Description**

Returns the remote tcp port from `skb`. 
**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**

```c
ipmib.ForwDatagrams
```

**Values**

- `op` value to be added to the counter (default value of 1)
- `skb` pointer to the struct sk_buff being acted on

**Description**

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

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 FragFails (equivalent to SNMP's MIB IPSTATS_MIB_FRAGFAILS)
probe::ipmib.FragOKs

probe::ipmib.FragOKs — Count datagram fragmented successfully

Synopsis

ipmib.FragOKs

Values

<table>
<thead>
<tr>
<th>skb</th>
<th>pointer to the struct sk_buff being acted on</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>value to be added to the counter (default value of 1)</td>
</tr>
</tbody>
</table>

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 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

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 InAddrErrors (equivalent to SNMP’s MIB IPSTATS_MIB_INADDRERRORS)
probe::ipmib.InDiscards

probe::ipmib.InDiscards — Count discarded inbound packets

Synopsis

ipmib.InDiscards

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 it 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

$skb$  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 $In\_No\_Routes$ (equivalent to SNMP's MIB IPSTATS\_MIB\_IN\_NOROUTES)
probe::ipmib.InReceives

probe::ipmib.InReceives — Count an arriving packet

Synopsis

ipmib.InReceives

Values

- skb: pointer to the struct sk_buff being acted on
- op: value to be added to the counter (default value of 1)

Description

The packet pointed to by skb is filtered by the function ipmib_filter_key. If the packet passes the filter is 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

\textit{skb} \quad \text{pointer to the struct sk_buff being acted on}

\textit{op} \quad \text{value to be added to the counter (default value of 1)}

Description

The packet pointed to by \textit{skb} is filtered by the function \texttt{ipmib\_filter\_key}. If the packet passes the filter is is counted in the global \textit{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

<table>
<thead>
<tr>
<th>op</th>
<th>value to be added to the counter (default value of 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>skb</td>
<td>pointer to the struct sk_buff being acted on</td>
</tr>
</tbody>
</table>

Description

The packet pointed to by skb is filtered by the function ipmib_filter_key. If the packet passes the filter it 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 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

skb  pointer to the struct sk_buff being acted on
op   value to be added to the counter (default value of 1)

Description

The packet pointed to by skb is filtered by the function ipmib_filter_key. If the packet passes the filter is is counted in the global ReasmTimeout (equivalent to SNMP’s MIB 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 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

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 it 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

\emph{op} Value to be added to the counter (default value of 1)

\emph{sk} Pointer to the struct sock being acted on

Description

The packet pointed to by \emph{skb} is filtered by the function \emph{linuxmib_filter_key}. If the packet passes the filter is is counted in the global \emph{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 it 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**

```c
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 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 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sk</td>
<td>pointer to the struct sock being acted on</td>
</tr>
<tr>
<td>op</td>
<td>value to be added to the counter (default value of 1)</td>
</tr>
</tbody>
</table>

Description

The packet pointed to by skb is filtered by the function tcpmib_filter_key. If the packet passes the filter is is counted in the global 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

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

sk  pointer to the struct sock being acted on

Description

The packet pointed to by skb is filtered by the function tcpmib_filter_key. If the packet passes the filter is is counted in the global 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 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 it 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)

sk  pointer to the struct sock being acted on

Description

The packet pointed to by skb is filtered by the function tcpmib_filter_key. If the packet passes the filter is is counted in the global 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

new_pid  The PID of the newly created process
new_tid  The TID of the newly created task

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

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

<table>
<thead>
<tr>
<th>name</th>
<th>Name of the system call (&quot;execve&quot;) (SystemTap v2.5+)</th>
</tr>
</thead>
<tbody>
<tr>
<td>errno</td>
<td>The error number resulting from the exec</td>
</tr>
<tr>
<td>success</td>
<td>A boolean indicating whether the exec was successful</td>
</tr>
<tr>
<td>retstr</td>
<td>A string representation of errno (SystemTap v2.5+)</td>
</tr>
</tbody>
</table>

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

\textit{code} \quad \text{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

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task</td>
<td>A task handle to the process being released</td>
</tr>
<tr>
<td>released_pid</td>
<td>PID of the process being released</td>
</tr>
<tr>
<td>pid</td>
<td>Same as released_pid for compatibility (deprecated)</td>
</tr>
<tr>
<td>released_tid</td>
<td>TID of the task being released</td>
</tr>
</tbody>
</table>

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

\[
sa\_flags\_str:\text{string}(sa\_flags:\text{long})
\]

Arguments

\[
\text{sa\_flags} \quad \text{the set of flags to convert to string.}
\]
**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

- **pid_name**: Name of the process receiving the signal
- **sig**: The number of the signal
- **sig_name**: A string representation of the signal
- **sig_pid**: The PID of the process receiving the signal
probe::signal.check_ignored.return

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

Synopsis

signal.check_ignored.return

Values

retstr    Return value as a string
name      Name of the probe point
probe::signal.checkperm

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

Synopsis

signal.checkperm

Values

<table>
<thead>
<tr>
<th>name</th>
<th>Name of the probe point</th>
</tr>
</thead>
<tbody>
<tr>
<td>sig_name</td>
<td>A string representation of the signal</td>
</tr>
<tr>
<td>sig_pid</td>
<td>The PID of the process receiving the signal</td>
</tr>
<tr>
<td>si_code</td>
<td>Indicates the signal type</td>
</tr>
<tr>
<td>sinfo</td>
<td>The address of the siginfo structure</td>
</tr>
<tr>
<td>pid_name</td>
<td>Name of the process receiving the signal</td>
</tr>
<tr>
<td>task</td>
<td>A task handle to the signal recipient</td>
</tr>
<tr>
<td>sig</td>
<td>The number of the signal</td>
</tr>
</tbody>
</table>
probe::signal.checkperm.return

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

Synopsis

signal.checkperm.return

Values

\( retstr \)  Return value as a string
\( name \)    Name of the probe point
probe::signal.do_action

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

Synopsis

signal.do_action

Values

<table>
<thead>
<tr>
<th>sig_name</th>
<th>A string representation of the signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the probe point</td>
</tr>
<tr>
<td>sa_handler</td>
<td>The new handler of the signal</td>
</tr>
<tr>
<td>sa_mask</td>
<td>The new mask of the signal</td>
</tr>
<tr>
<td>oldsigact_addr</td>
<td>The address of the old sigaction struct associated with the signal</td>
</tr>
<tr>
<td>sig</td>
<td>The signal to be examined/changed</td>
</tr>
<tr>
<td>sigact_addr</td>
<td>The address of the new sigaction struct associated with the signal</td>
</tr>
</tbody>
</table>
probe::signal.do_action.return

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

Synopsis

signal.do_action.return

Values

\textit{retstr} \quad \text{Return value as a string}
\textit{name} \quad \text{Name of the probe point}
probe::signal.flush

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

Synopsis

signal.flush

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sig_pid</td>
<td>The PID of the process associated with the task performing the flush</td>
</tr>
<tr>
<td>name</td>
<td>Name of the probe point</td>
</tr>
<tr>
<td>task</td>
<td>The task handler of the process performing the flush</td>
</tr>
<tr>
<td>pid_name</td>
<td>The name of the process associated with the task performing the flush</td>
</tr>
</tbody>
</table>
probe::signal.force_segv

probe::signal.force_segv — Forcing send of SIGSEGV

Synopsis

signal.force_segv

Values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sig</td>
<td>The number of the signal</td>
</tr>
<tr>
<td>pid_name</td>
<td>Name of the process receiving the signal</td>
</tr>
<tr>
<td>name</td>
<td>Name of the probe point</td>
</tr>
<tr>
<td>sig_pid</td>
<td>The PID of the process receiving the signal</td>
</tr>
<tr>
<td>sig_name</td>
<td>A string representation of the signal</td>
</tr>
</tbody>
</table>
probe::signal.force_segv.return

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

Synopsis

signal.force_segv.return

Values

<table>
<thead>
<tr>
<th>name</th>
<th>Name of the probe point</th>
</tr>
</thead>
<tbody>
<tr>
<td>retstr</td>
<td>Return value as a string</td>
</tr>
</tbody>
</table>
probe::signal.handle

probe::signal.handle — Signal handler being invoked

Synopsis

signal.handle

Values

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

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

**Synopsis**

```plaintext
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

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

Description

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

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

Synopsis

signal.pending.return

Values

<table>
<thead>
<tr>
<th>name</th>
<th>Name of the probe point</th>
</tr>
</thead>
<tbody>
<tr>
<td>retstr</td>
<td>Return value as a string</td>
</tr>
</tbody>
</table>
**probe::signal.procmask**

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

**Synopsis**

signal.procmask

**Values**

- **name**: Name of the probe point
- **how**: 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.
- **oldsigset_addr**: The old address of the signal set (sigset_t)
- **sigset**: The actual value to be set for sigset_t (correct?)
- **sigset_addr**: The address of the signal set (sigset_t) to be implemented
**probe::signal.procmask.return**

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

**Synopsis**

signal.procmask.return

**Values**

<table>
<thead>
<tr>
<th>retstr</th>
<th>Return value as a string</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the probe point</td>
</tr>
</tbody>
</table>
**probe::signal.send**

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

**Synopsis**

`signal.send`

**Values**

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

**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**

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

**Context**

The signal's sender. (correct?)

**Description**

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

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

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

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

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

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

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

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

Synopsis

signal.send_sig_queue

Values

<table>
<thead>
<tr>
<th>name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sigqueue_addr</td>
<td>The address of the signal queue</td>
</tr>
<tr>
<td>sig_pid</td>
<td>The PID of the process to which the signal is queued</td>
</tr>
<tr>
<td>sig_name</td>
<td>A string representation of the signal</td>
</tr>
<tr>
<td>name</td>
<td>Name of the probe point</td>
</tr>
<tr>
<td>pid_name</td>
<td>Name of the process to which the signal is queued</td>
</tr>
<tr>
<td>sig</td>
<td>The queued signal</td>
</tr>
</tbody>
</table>
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

\texttt{retstr} \hspace{5mm} \text{Return value as a string}

\texttt{name} \hspace{5mm} \text{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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tgid</td>
<td>The thread group ID of the thread receiving the kill signal</td>
</tr>
<tr>
<td>sig</td>
<td>The specific kill signal sent to the process</td>
</tr>
<tr>
<td>pid_name</td>
<td>The name of the signal recipient</td>
</tr>
<tr>
<td>task</td>
<td>A task handle to the signal recipient</td>
</tr>
<tr>
<td>name</td>
<td>Name of the probe point</td>
</tr>
<tr>
<td>sig_pid</td>
<td>The PID of the thread receiving the kill signal</td>
</tr>
<tr>
<td>sig_name</td>
<td>A string representation of the signal</td>
</tr>
</tbody>
</table>

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

- **name**: Name of the probe point
- **retstr**: The return value to either __group_send_sig_info,
probe::signal.sys_tkill

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

Synopsis

signal.sys_tkill

Values

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task</td>
<td>A task handle to the signal recipient</td>
</tr>
<tr>
<td>pid_name</td>
<td>The name of the signal recipient</td>
</tr>
<tr>
<td>sig</td>
<td>The specific signal sent to the process</td>
</tr>
<tr>
<td>sig_pid</td>
<td>The PID of the process receiving the kill signal</td>
</tr>
<tr>
<td>sig_name</td>
<td>A string representation of the signal</td>
</tr>
<tr>
<td>name</td>
<td>Name of the probe point</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>name</th>
<th>Name of the probe point</th>
</tr>
</thead>
<tbody>
<tr>
<td>sig_name</td>
<td>A string representation of the signal</td>
</tr>
<tr>
<td>sig_pid</td>
<td>The PID of the process receiving the signal</td>
</tr>
<tr>
<td>pid_name</td>
<td>The name of the signal recipient</td>
</tr>
<tr>
<td>task</td>
<td>A task handle to the signal recipient</td>
</tr>
<tr>
<td>sig</td>
<td>The specific signal sent to the process</td>
</tr>
</tbody>
</table>
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

*name*  Name of the probe point

*retstr*  The return value to either __group_send_sig_info,
probe::signal.wakeup

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

Synopsis

signal.wakeup

Values

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

format Variable to determine return type base value
ret Return value (typically $return)

Description

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

Note that this function is preferred over returnstr.
function::returnstr

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

Synopsis

\[ \text{returnstr::string}(\text{format}: \text{long}) \]

Arguments

\[ \text{format} \quad \text{Variable to determine return type base value} \]

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

```c
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

\textit{task} \hspace{1em} task_struct pointer.

\textit{file} \hspace{1em} Pointer to “struct file”.

Description

Returns the full directory 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(path:)
```

**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

task task_struct pointer.
dentry dirent pointer.
vfsmnt vfsmnt pointer.

Description

Returns the full dirent name (full path to the root), like the kernel d_path function.
Chapter 22. Logging Tapset

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

function::assert — evaluate assertion

Synopsis

assert(expression:,msg:)

Arguments

expression The expression to evaluate
msg The formatted message string

Description

This function checks the expression and aborts the current running probe if expression evaluates to zero. Uses error and may be caught by try{} catch{}. 
function::error

function::error — Send an error message

Synopsis

\[ \text{error(msg:string)} \]

Arguments

\[ msg \quad \text{The formatted message string} \]

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_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

\[ \text{kernel_int:long}(\text{addr:long}) \]

Arguments

\[ \text{addr} \quad \text{The kernel address to retrieve the int from} \]

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

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**

    kernel_string:string(addr:long)

**Arguments**

    addr    The kernel address to retrieve the string from

**Description**

This function returns the null terminated C string from a given kernel memory address. Reports an error on string copy fault.
function::kernel_string2

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

Synopsis

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

Arguments

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

Description

This function returns the null terminated C string from a given kernel memory address. Reports the given error message on string copy fault.
function::kernel_string2Utf16

function::kernel_string2Utf16 — Retrieves UTF-16 string from kernel memory with alternative error string

Synopsis

kernel_string2Utf16:string(addr:long, err_msg:string)

Arguments

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

Description

This function returns a null terminated UTF-8 string converted from the UTF-16 string at a given kernel memory address. Reports the given error message on string copy fault or conversion error.
function::kernel_string2_utf32

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

Synopsis

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

Arguments

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

Description

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

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

Synopsis

\[
\text{kernel_string_n:} \text{string}(\text{addr:} \text{long}, \text{n:} \text{long})
\]

Arguments

\begin{itemize}
  \item \textit{addr} \quad \text{The kernel address to retrieve the string from}
  \item \textit{n} \quad \text{The maximum length of the string (if not null terminated)}
\end{itemize}

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.
String and data retrieving
functions Tapset

function::kernel_string_utf16

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

Synopsis

kernel_string_utf16:string(addr:long)

Arguments

addr   The kernel address to retrieve the string from

Description

This function returns a null terminated UTF-8 string converted from the UTF-16 string at a given kernel memory address. Reports an error on string copy fault or conversion error.
function::kernel_string_utf32

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

Synopsis

    kernel_string_utf32::string(addr:long)

Arguments

    addr  The kernel address to retrieve the string from

Description

    This function returns a null terminated UTF-8 string converted from the UTF-32 string at a given kernel memory address. Reports an error on string copy fault or conversion error.
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_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 and warns (but does not abort) about the failure.
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**

Retrieves a 16-bit integer value stored in user space

**Synopsis**

```c
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_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_int64

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

Synopsis

user_int64:long(addr:long)

Arguments

\textit{addr} \quad \text{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_int8

function::user_int8 — Retrieves a 8-bit integer value stored in user space

Synopsis

\texttt{user\_int8:long(addr:long)}

Arguments

\textit{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_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 and warns (but does not abort) about the failure.
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_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 and warns (but does not abort) about the failure. Note that the size of the long depends on the architecture of the current user space task (for those architectures that support both 64/32 bit compat tasks).
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_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 and warns (but does not abort) about the failure.
function::user_string

function::user_string — Retrieves string from user space

Synopsis

user_string:string(addr:long)

Arguments

addr the user space address to retrieve the string from

Description

Returns the null terminated C string from a given user space memory address. Reports an error on the rare cases when userspace data is not accessible.
function::user_string2

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

Synopsis

\[
\text{user_string2:} \text{string}(\text{addr: long, err_msg: string})
\]

Arguments

- \textit{addr} — the user space address to retrieve the string from
- \textit{err_msg} — the error message to return when data isn't available

Description

Returns the null terminated C string from a given user space memory address. Reports the given error message on the rare cases when userspace data is not accessible.
function::user_string2_n_warn

function::user_string2_n_warn — Retrieves string from user space with alternative warning string

Synopsis

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

Arguments

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

Description

Returns up to n characters of a C string from a given user space memory address. Reports the given warning message on the rare cases when userspace data is not accessible and warns (but does not abort) about the failure.
function::user_string2_utf16

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

Synopsis

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

Arguments

addr  The user address to retrieve the string from
err_msg  The error message to return when data isn't available

Description

This function returns a null terminated UTF-8 string converted from the UTF-16 string at a given user memory address. Reports the given error message on string copy fault or conversion error.
function::user_string2_utf32

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

Synopsis

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

Arguments

addr The user address to retrieve the string from
err_msg The error message to return when data isn't available

Description

This function returns a null terminated UTF-8 string converted from the UTF-32 string at a given user memory address. Reports the given error message on string copy fault or conversion error.
function::user_string2_warn

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

Synopsis

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

Arguments

 addr              the user space address to retrieve the string from
 warn_msg          the warning message to return when data isn't available

Description

Returns the null terminated C string from a given user space memory address. Reports the given warning message on the rare cases when userspace data is not accessible and warns (but does not abort) about the failure.
function::user_string_n

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

Synopsis

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

Arguments

addr the user space address to retrieve the string from

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

Description

Returns the C string of a maximum given length from a given user space address. Reports an error on the rare cases when userspace data is not accessible at the given address.
function::user_string_n2

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

Synopsis

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

Arguments

addr the user space address to retrieve the string from

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

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

Description

Returns the C string of a maximum given length from a given user space address. Returns the given error message string on the rare cases when userspace data is not accessible at the given address.
function::user_string_n2_quoted

    function::user_string_n2_quoted — Retrieves and quotes string from user space

Synopsis

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

Arguments

    addr        the user space address to retrieve the string from
    inlen       the maximum length of the string to read (if not null terminated)
    outlen      the maximum length of the output string

Description

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

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

Synopsis

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

Arguments

addr the user space address to retrieve the string from

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

Description

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

function::user_string_n_warn — Retrieves string from user space

**Synopsis**

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

**Arguments**

- **addr** the user space address to retrieve the string from
- **n** the maximum length of the string (if not null terminated)

**Description**

Returns up to n characters of a C string from a given user space memory address. Reports “<unknown>” on the rare cases when userspace data is not accessible and warns (but does not abort) about the failure.
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_utf16

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

Synopsis

user_string_utf16::string(addr:long)

Arguments

addr The user address to retrieve the string from

Description

This function returns a null terminated UTF-8 string converted from the UTF-16 string at a given user memory address. Reports an error on string copy fault or conversion error.
function::user_string_utf32

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

Synopsis

user_string_utf32:string(addr:long)

Arguments

addr The user address to retrieve the string from

Description

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

function::user_string_warn — Retrieves string from user space

Synopsis

user_string_warn:string(addr:long)

Arguments

 addr the user space address to retrieve the string from

Description

Returns the null terminated C string from a given user space memory address. Reports "" on the rare cases when userspace data is not accessible and warns (but does not abort) about the failure.
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_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_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_uint8

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

Synopsis

user_uint8:long(addr:long)

Arguments

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

Description

Returns the unsigned 8-bit integer value from a given user space address. Returns zero when user space data is not accessible.
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_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 and warns (but does not abort) about the failure. Note that the size of the unsigned long depends on the architecture of the current user space task (for those architectures that support both 64/32 bit compat tasks).
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_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 and warns (but does not abort) about the failure.
Chapter 26. String and data writing functions Tapset

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

```c
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

<table>
<thead>
<tr>
<th>addr</th>
<th>The kernel address to write the string to</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>The maximum length of the string</td>
</tr>
<tr>
<td>val</td>
<td>The string which is to be written</td>
</tr>
</tbody>
</table>

Description

Writes the given string up to a maximum given length to a given kernel memory address. Reports an error on string copy fault. Requires the use of guru mode (-g).
Chapter 27. Guru tapsets

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

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

Synopsis

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

Arguments

$s1$  string to search in
$s2$  substring to find

Description

This function returns 1 if string $s1$ contains $s2$, otherwise zero.
function::str_replace

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

Synopsis

\[
\text{str\_replace: string(prnt\_str:string, srch\_str:string, rplc\_str:string)}
\]

Arguments

- **prnt\_str**: the string to search and replace in
- **srch\_str**: the substring which is used to search in **prnt\_str** string
- **rplc\_str**: the substring which is used to replace **srch\_str**

Description

This function returns the given string with substrings replaced.
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.
A collection of standard string functions

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

\texttt{substr::string(str::string, start::long, length::long)}

Arguments

\begin{itemize}
\item \textit{str} \quad the string to take a substring from
\item \textit{start} \quad starting position of the extracted string (first character is 0)
\item \textit{length} \quad length of string to return
\end{itemize}

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.
A collection of standard
string functions

**function::text_strn**

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

**Synopsis**

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

**Arguments**

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

**Description**

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

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

Synopsis

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

Arguments

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

delim set of characters that delimit the tokens

Description

This function returns the next non-empty token in the given input string, where the tokens are delimited by characters in the delim string. If the input string is non-empty, it returns the first token. If the input string is empty, it returns the next token in the string passed in the previous call to tokenize. If no delimiter is found, the entire remaining input string is returned. It returns empty when no more tokens are available.
Chapter 29. Utility functions for using ansi control chars in logs

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

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.
Utility functions for using ansi control chars in logs

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.
Utility functions for using ansi control chars in logs

**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.
Utility functions for using ansi control chars in logs

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.
Utility functions for using
ansi control chars in logs

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.
Utility functions for using ansi control chars in logs

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

ansi_set_color(fg:long)

Arguments

fg  Foreground color to set.

Description

Sends ansi code for Select Graphic Rendition mode for the given forground color. Black (30), Blue (34), Green (32), Cyan (36), Red (31), Purple (35), Brown (33), Light Gray (37).
function::ansi_set_color2

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

Synopsis

ansi_set_color2(fg:long, bg:long)

Arguments

fg  Foreground color to set.
bg  Background color to set.

Description

Sends ansi code for Select Graphic Rendition mode for the given foreground color, Black (30), Blue (34), Green (32), Cyan (36), Red (31), Purple (35), Brown (33), Light Gray (37) and the given background color, Black (40), Red (41), Green (42), Yellow (43), Blue (44), Magenta (45), Cyan (46), White (47).
function::ansi_set_color3

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

Synopsis

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

Arguments

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

Description

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

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

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

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

Description

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

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

Description

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

source_path the path of the .c source file
module_path the path of the .ko kernel module 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

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

**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

ret the return value from posix_spawn
pid the pid of the spawned process

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

- **len** the length (in bytes) of the data blob
- **data** a ptr to a binary blob of data sent as the control message
- **type** type of message being send; defined in runtime/transport/transport_msgs.h

Description

Fires at the beginning of the send_request function.
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**

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

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

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

- **size**: release pages
- **__page**: the address of page
- **page_index**: offset within mapping, can used a page identifier and position identifier in the page frame
- **ino**: inode number
- **dev**: device identifier

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

\begin{itemize}
  \item \texttt{page\_flag} \hfill page flags
  \item \texttt{\_\_page} \hfill the address of page
\end{itemize}

Description

This probe attaches to the generic \texttt{\_\_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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>__page</td>
<td>the address of page</td>
</tr>
<tr>
<td>size</td>
<td>write bytes</td>
</tr>
<tr>
<td>page_index</td>
<td>offset within mapping, can used a page identifier and position identifier in the page frame</td>
</tr>
<tr>
<td>dev</td>
<td>device identifier</td>
</tr>
<tr>
<td>ino</td>
<td>inode number</td>
</tr>
<tr>
<td>offset</td>
<td>start address of this write operation</td>
</tr>
<tr>
<td>to</td>
<td>end address of this write operation</td>
</tr>
</tbody>
</table>

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

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos</td>
<td>current position of file</td>
</tr>
<tr>
<td>cache_valid</td>
<td>cache related bit mask flag</td>
</tr>
<tr>
<td>attrtimeo</td>
<td>how long the cached information is assumed to be valid. We need to revalidate the cached attrs for this inode if jiffies - read_cache_jiffies &gt; attrtimeo.</td>
</tr>
<tr>
<td>buf</td>
<td>the address of buf in user space</td>
</tr>
<tr>
<td>file_name</td>
<td>file name</td>
</tr>
<tr>
<td>cache_time</td>
<td>when we started read-caching this inode</td>
</tr>
<tr>
<td>count</td>
<td>read bytes</td>
</tr>
<tr>
<td>dev</td>
<td>device identifier</td>
</tr>
<tr>
<td>ino</td>
<td>inode number</td>
</tr>
<tr>
<td>parent_name</td>
<td>parent dir name</td>
</tr>
</tbody>
</table>
probe::nfs.fop.aio_write

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

Synopsis

nfs.fop.aio_write

Values

- **pos**: offset of the file
- **ino**: inode number
- **parent_name**: parent dir name
- **file_name**: file name
- **dev**: device identifier
- **count**: read bytes
- **buf**: the address of buf in user space
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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>file mode</td>
</tr>
<tr>
<td>ino</td>
<td>inode number</td>
</tr>
<tr>
<td>ndirty</td>
<td>number of dirty page</td>
</tr>
<tr>
<td>dev</td>
<td>device identifier</td>
</tr>
</tbody>
</table>
**probe::nfs.fop.fsync**

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

**Synopsis**

```
nfs.fop.fsync
```

**Values**

- `dev`: device identifier
- `ndirty`: number of dirty pages
- `ino`: inode number
**probe::nfs.fop.llseek**

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

**Synopsis**

nfs.fop.llseek

**Values**

- **offset**: the offset of the file will be repositioned
- **whence**: the position to seek from
- **whence_str**: symbolic string representation of the position to seek from
- **ino**: inode number
- **dev**: device identifier
probe::nfs.fop.lock

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

Synopsis

nfs.fop.lock

Values

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fl_end</td>
<td>ending offset of locked region</td>
</tr>
<tr>
<td>fl_flag</td>
<td>lock flags</td>
</tr>
<tr>
<td>ino</td>
<td>inode number</td>
</tr>
<tr>
<td>fl_start</td>
<td>starting offset of locked region</td>
</tr>
<tr>
<td>i_mode</td>
<td>file type and access rights</td>
</tr>
<tr>
<td>fl_type</td>
<td>lock type</td>
</tr>
<tr>
<td>dev</td>
<td>device identifier</td>
</tr>
<tr>
<td>cmd</td>
<td>cmd arguments</td>
</tr>
</tbody>
</table>
**probe::nfs.fop.mmap**

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

### Synopsis

nfs.fop.mmap

### Values

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

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

Synopsis

nfs.fop.open

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flag</td>
<td>file flag</td>
</tr>
<tr>
<td>_i_size</td>
<td>file length in bytes</td>
</tr>
<tr>
<td>dev</td>
<td>device identifier</td>
</tr>
<tr>
<td>file_name</td>
<td>file name</td>
</tr>
<tr>
<td>ino</td>
<td>inode number</td>
</tr>
</tbody>
</table>
**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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ino</td>
<td>inode number</td>
</tr>
<tr>
<td>parent_name</td>
<td>parent dir name</td>
</tr>
<tr>
<td>file_name</td>
<td>file name</td>
</tr>
<tr>
<td>cache_time</td>
<td>when we started read-caching this inode</td>
</tr>
<tr>
<td>count</td>
<td>read bytes</td>
</tr>
<tr>
<td>dev</td>
<td>device identifier</td>
</tr>
<tr>
<td>attrtimeo</td>
<td>how long the cached information is assumed to be valid. We need to revalidate the cached attrs for this inode if jiffies - read_cache_jiffies &gt; attrtimeo.</td>
</tr>
<tr>
<td>cache_valid</td>
<td>cache related bit mask flag</td>
</tr>
<tr>
<td>pos</td>
<td>current position of file</td>
</tr>
</tbody>
</table>
probe::nfs.fop.release

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

Synopsis

nfs.fop.release

Values

  \textit{mode} file mode
  \textit{dev} device identifier
  \textit{ino} inode number
probe::nfs.fop.sendfile

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

Synopsis

nfs.fop.sendfile

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ino</td>
<td>inode number</td>
</tr>
<tr>
<td>cache_time</td>
<td>when we started read-caching this inode</td>
</tr>
<tr>
<td>dev</td>
<td>device identifier</td>
</tr>
<tr>
<td>count</td>
<td>read bytes</td>
</tr>
<tr>
<td>cache_valid</td>
<td>cache related bit mask flag</td>
</tr>
<tr>
<td>attrtimeo</td>
<td>how long the cached information is assumed to be valid. We need to revalidate the cached attrs for this inode if jiffies - read_cache_jiffies &gt; attrtimeo.</td>
</tr>
<tr>
<td>ppos</td>
<td>current position of file</td>
</tr>
</tbody>
</table>
probe::nfs.fop.write

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

Synopsis

nfs.fop.write

Values

\texttt{devname} \hspace{1em} \text{block device name}

Description

SystemTap uses the \texttt{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

<table>
<thead>
<tr>
<th>Pos</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos</td>
<td>offset of the file</td>
</tr>
<tr>
<td>ino</td>
<td>inode number</td>
</tr>
<tr>
<td>parent_name</td>
<td>parent dir name</td>
</tr>
<tr>
<td>file_name</td>
<td>file name</td>
</tr>
<tr>
<td>dev</td>
<td>device identifier</td>
</tr>
<tr>
<td>count</td>
<td>read bytes</td>
</tr>
</tbody>
</table>
**probe::nfs.proc.commit**

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

**Synopsis**

nfs.proc.commit

**Values**

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

**Description**

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

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

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

Synopsis

nfs.proc.commit_done

Values

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server_ip</td>
<td>IP address of server</td>
</tr>
<tr>
<td>status</td>
<td>result of last operation</td>
</tr>
<tr>
<td>count</td>
<td>number of bytes committed</td>
</tr>
<tr>
<td>version</td>
<td>NFS version</td>
</tr>
<tr>
<td>valid</td>
<td>fattr-&gt;valid, indicates which fields are valid</td>
</tr>
<tr>
<td>timestamp</td>
<td>V4 timestamp, which is used for lease renewal</td>
</tr>
<tr>
<td>prot</td>
<td>transfer protocol</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server_ip</td>
<td>IP address of server</td>
</tr>
<tr>
<td>count</td>
<td>bytes in this commit</td>
</tr>
<tr>
<td>version</td>
<td>NFS version</td>
</tr>
<tr>
<td>bitmask1</td>
<td>V4 bitmask representing the set of attributes supported on this filesystem</td>
</tr>
<tr>
<td>offset</td>
<td>the file offset</td>
</tr>
<tr>
<td>bitmask0</td>
<td>V4 bitmask representing the set of attributes supported on this filesystem</td>
</tr>
<tr>
<td>prot</td>
<td>transfer protocol</td>
</tr>
<tr>
<td>size</td>
<td>bytes in this commit</td>
</tr>
</tbody>
</table>

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.
probes::nfs.proc.create

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

Synopsis

nfs.proc.create

Values

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

- `version` : NFS version
- `bitmask1` : V4 bitmask representing the set of attributes supported on this filesystem
- `bitmask0` : V4 bitmask representing the set of attributes supported on this filesystem
- `prot` : transfer protocol
- `filename` : the name of file which client opens/searches on server
- `server_ip` : IP address of server
- `name_len` : the length of file name
probe::nfs.proc.open

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

Synopsis

nfs.proc.open

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>file mode</td>
</tr>
<tr>
<td>version</td>
<td>NFS version (the function is used for all NFS version)</td>
</tr>
<tr>
<td>server_ip</td>
<td>IP address of server</td>
</tr>
<tr>
<td>filename</td>
<td>file name</td>
</tr>
<tr>
<td>flag</td>
<td>file flag</td>
</tr>
<tr>
<td>prot</td>
<td>transfer protocol</td>
</tr>
</tbody>
</table>

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

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

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

- **server_ip**: IP address of server
- **count**: number of bytes read
- **status**: result of last operation
- **version**: NFS version
- **timestamp**: V4 timestamp, which is used for lease renewal
- **prot**: 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

- **size**: read bytes in this execution
- **prot**: transfer protocol
- **offset**: the file offset
- **version**: NFS version
- **count**: read bytes in this execution
- **server_ip**: IP address of server

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>file mode</td>
</tr>
<tr>
<td>version</td>
<td>NFS version (the function is used for all NFS version)</td>
</tr>
<tr>
<td>server_ip</td>
<td>IP address of server</td>
</tr>
<tr>
<td>filename</td>
<td>file name</td>
</tr>
<tr>
<td>flag</td>
<td>file flag</td>
</tr>
<tr>
<td>prot</td>
<td>transfer protocol</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filelen</td>
<td>length of file name</td>
</tr>
<tr>
<td>fh</td>
<td>file handle of parent dir</td>
</tr>
<tr>
<td>prot</td>
<td>transfer protocol</td>
</tr>
<tr>
<td>version</td>
<td>NFS version (the function is used for all NFS version)</td>
</tr>
<tr>
<td>server_ip</td>
<td>IP address of server</td>
</tr>
<tr>
<td>filename</td>
<td>file name</td>
</tr>
</tbody>
</table>
probe::nfs.proc.rename

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

Synopsis

nfs.proc.rename

Values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>NFS version (the function is used for all NFS version)</td>
</tr>
<tr>
<td>server_ip</td>
<td>IP address of server</td>
</tr>
<tr>
<td>old_fh</td>
<td>file handle of old parent dir</td>
</tr>
<tr>
<td>new_fh</td>
<td>file handle of new parent dir</td>
</tr>
<tr>
<td>old_name</td>
<td>old file name</td>
</tr>
<tr>
<td>new_name</td>
<td>new file name</td>
</tr>
<tr>
<td>prot</td>
<td>transfer protocol</td>
</tr>
<tr>
<td>new_filelen</td>
<td>length of new file name</td>
</tr>
<tr>
<td>old_filelen</td>
<td>length of old file name</td>
</tr>
</tbody>
</table>
probe::nfs.proc.rename_done

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

Synopsis

nfs.proc.rename_done

Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prot</td>
<td>transfer protocol</td>
</tr>
<tr>
<td>old_fh</td>
<td>file handle of old parent dir</td>
</tr>
<tr>
<td>server_ip</td>
<td>IP address of server</td>
</tr>
<tr>
<td>new_fh</td>
<td>file handle of new parent dir</td>
</tr>
<tr>
<td>status</td>
<td>result of last operation</td>
</tr>
<tr>
<td>version</td>
<td>NFS version</td>
</tr>
<tr>
<td>timestamp</td>
<td>V4 timestamp, which is used for lease renewal</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>NFS version</td>
</tr>
<tr>
<td>server_ip</td>
<td>IP address of server</td>
</tr>
<tr>
<td>prot</td>
<td>transfer protocol</td>
</tr>
<tr>
<td>fh</td>
<td>file handle of parent dir</td>
</tr>
</tbody>
</table>

Description

The rename_setup function is used to setup a rename RPC task. Is 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**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>read bytes in this execution</td>
</tr>
<tr>
<td>prot</td>
<td>transfer protocol</td>
</tr>
<tr>
<td>flags</td>
<td>used to set task-&gt;tk_flags in rpc_init_task function</td>
</tr>
<tr>
<td>server_ip</td>
<td>IP address of server</td>
</tr>
<tr>
<td>offset</td>
<td>the file offset</td>
</tr>
<tr>
<td>bitmask0</td>
<td>V4 bitmask representing the set of attributes supported on this filesystem</td>
</tr>
<tr>
<td>bitmask1</td>
<td>V4 bitmask representing the set of attributes supported on this filesystem</td>
</tr>
<tr>
<td>version</td>
<td>NFS version</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prot</td>
<td>transfer protocol</td>
</tr>
<tr>
<td>server_ip</td>
<td>IP address of server</td>
</tr>
<tr>
<td>count</td>
<td>number of bytes written</td>
</tr>
<tr>
<td>status</td>
<td>result of last operation</td>
</tr>
<tr>
<td>valid</td>
<td>fattr-&gt;valid, indicates which fields are valid</td>
</tr>
<tr>
<td>version</td>
<td>NFS version</td>
</tr>
<tr>
<td>timestamp</td>
<td>V4 timestamp, which is used for lease renewal</td>
</tr>
</tbody>
</table>

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**

- **how**: 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)
- **prot**: transfer protocol
- **size**: bytes written in this execution
- **version**: NFS version
- **bitmask0**: V4 bitmask representing the set of attributes supported on this filesystem
- **offset**: file offset
- **bitmask1**: V4 bitmask representing the set of attributes supported on this filesystem
- **server_ip**: IP address of server
- **count**: bytes written in this execution

**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

filename 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

<table>
<thead>
<tr>
<th>count</th>
<th>read bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>fh</td>
<td>file handle (the first part is the length of the file handle)</td>
</tr>
<tr>
<td>client_ip</td>
<td>the ip address of client</td>
</tr>
<tr>
<td>offset</td>
<td>the offset of file</td>
</tr>
<tr>
<td>size</td>
<td>read bytes</td>
</tr>
<tr>
<td>flag</td>
<td>indicates whether this execution is a sync operation</td>
</tr>
</tbody>
</table>
probe::nfsd.create

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

Synopsis

nfsd.create

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>file name</td>
</tr>
<tr>
<td>client_ip</td>
<td>the ip address of client</td>
</tr>
<tr>
<td>type</td>
<td>file type(regular,dir,device,fifo ...)</td>
</tr>
<tr>
<td>iap_valid</td>
<td>Attribute flags</td>
</tr>
<tr>
<td>fh</td>
<td>file handle (the first part is the length of the file handle)</td>
</tr>
<tr>
<td>iap_mode</td>
<td>file access mode</td>
</tr>
<tr>
<td>filelen</td>
<td>the length of file name</td>
</tr>
</tbody>
</table>

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

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

iap_valid
    Attribute flags

client_ip
    the ip address of client

filename
    file name

verifier
    file attributes (atime,mtime,mode). It's used to reset file attributes for CREATE_EXCLUSIVE

truncp
    trunp arguments, indicates if the file should be truncate

createmode
    create mode. The possible values could be: NFS3_CREATE_EXCLUSIVE, NFS3_CREATE_UNCHECKED, or NFS3_CREATE_GUARDED

filelen
    the length of file name

iap_mode
    file access mode

Description

This probe points 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>proto</td>
<td>transfer protocol</td>
</tr>
<tr>
<td>client_ip</td>
<td>the ip address of client</td>
</tr>
<tr>
<td>version</td>
<td>nfs version</td>
</tr>
<tr>
<td>xid</td>
<td>transmission id</td>
</tr>
<tr>
<td>prog</td>
<td>program number</td>
</tr>
<tr>
<td>proc</td>
<td>procedure number</td>
</tr>
</tbody>
</table>
probe::nfsd.lookup

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

Synopsis

nfsd.lookup

Values

fh file handle of parent dir (the first part is the length of the file handle)
filename file name
client_ip the ip address of client
filelen the length of file name
## probe::nfsd.open

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

### Synopsis

```plaintext
nfsd.open
```

### Values

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>access</code></td>
<td>Indicates the type of open (read/write/commit/readdir...)</td>
</tr>
<tr>
<td><code>client_ip</code></td>
<td>The IP address of client</td>
</tr>
<tr>
<td><code>type</code></td>
<td>Type of file (regular file or dir)</td>
</tr>
<tr>
<td><code>fh</code></td>
<td>File handle (the first part is the length of the file handle)</td>
</tr>
</tbody>
</table>
probe::nfsd.proc.commit

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

Synopsis

nfsd.proc.commit

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uid</td>
<td>requester's user id</td>
</tr>
<tr>
<td>offset</td>
<td>the offset of file</td>
</tr>
<tr>
<td>size</td>
<td>read bytes</td>
</tr>
<tr>
<td>fh</td>
<td>file handle (the first part is the length of the file handle)</td>
</tr>
<tr>
<td>proto</td>
<td>transfer protocol</td>
</tr>
<tr>
<td>client_ip</td>
<td>the ip address of client</td>
</tr>
<tr>
<td>version</td>
<td>nfs version</td>
</tr>
<tr>
<td>gid</td>
<td>requester's group id</td>
</tr>
<tr>
<td>count</td>
<td>read bytes</td>
</tr>
</tbody>
</table>
**probe::nfsd.proc.create**

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

**Synopsis**

nfsd.proc.create

**Values**

- **filelen**: length of file name
- **uid**: requester's user id
- **gid**: requester's group id
- **version**: nfs version
- **client_ip**: the ip address of client
- **proto**: transfer protocol
- **filename**: file name
- **fh**: 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

- `filelen` the length of file name
- `uid` requester's user id
- `gid` requester's group id
- `version` nfs version
- `client_ip` the ip address of client
- `filename` file name
- `proto` transfer protocol
- `fh` file handle of parent dir (the first part is the length of the file handle)
probe::nfsd.proc.read

probe::nfsd.proc.read — NFS server reading file for client

Synopsis

nfsd.proc.read

Values

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uid</td>
<td>requester's user id</td>
</tr>
<tr>
<td>vlen</td>
<td>read blocks</td>
</tr>
<tr>
<td>size</td>
<td>read bytes</td>
</tr>
<tr>
<td>offset</td>
<td>the offset of file</td>
</tr>
<tr>
<td>vec</td>
<td>struct kvec, includes buf address in kernel address and length of each buffer</td>
</tr>
<tr>
<td>client_ip</td>
<td>the ip address of client</td>
</tr>
<tr>
<td>version</td>
<td>nfs version</td>
</tr>
<tr>
<td>proto</td>
<td>transfer protocol</td>
</tr>
<tr>
<td>fh</td>
<td>file handle (the first part is the length of the file handle)</td>
</tr>
<tr>
<td>count</td>
<td>read bytes</td>
</tr>
<tr>
<td>gid</td>
<td>requester's group id</td>
</tr>
</tbody>
</table>
probe::nfsd.proc.remove

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

Synopsis

nfsd.proc.remove

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gid</td>
<td>requester's group id</td>
</tr>
<tr>
<td>fh</td>
<td>file handle (the first part is the length of the file handle)</td>
</tr>
<tr>
<td>filename</td>
<td>file name</td>
</tr>
<tr>
<td>proto</td>
<td>transfer protocol</td>
</tr>
<tr>
<td>client_ip</td>
<td>the ip address of client</td>
</tr>
<tr>
<td>version</td>
<td>nfs version</td>
</tr>
<tr>
<td>filelen</td>
<td>length of file name</td>
</tr>
<tr>
<td>uid</td>
<td>requester's user id</td>
</tr>
</tbody>
</table>
probe::nfsd.proc.rename

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

Synopsis

nfsd.proc.rename

Values

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uid</td>
<td>requester's user id</td>
</tr>
<tr>
<td>client_ip</td>
<td>the ip address of client</td>
</tr>
<tr>
<td>filename</td>
<td>old file name</td>
</tr>
<tr>
<td>tfh</td>
<td>file handler of new path</td>
</tr>
<tr>
<td>fh</td>
<td>file handler of old path</td>
</tr>
<tr>
<td>tlen</td>
<td>length of new file name</td>
</tr>
<tr>
<td>flen</td>
<td>length of old file name</td>
</tr>
<tr>
<td>tname</td>
<td>new file name</td>
</tr>
<tr>
<td>gid</td>
<td>requester's group id</td>
</tr>
</tbody>
</table>
probe::nfsd.proc.write

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

Synopsis

nfsd.proc.write

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>nfs version</td>
</tr>
<tr>
<td>vec</td>
<td>struct kvec, includes buf address in kernel address and length of each buffer</td>
</tr>
<tr>
<td>client_ip</td>
<td>the ip address of client</td>
</tr>
<tr>
<td>proto</td>
<td>transfer protocol</td>
</tr>
<tr>
<td>fh</td>
<td>file handle (the first part is the length of the file handle)</td>
</tr>
<tr>
<td>stable</td>
<td>argp-&gt;stable</td>
</tr>
<tr>
<td>count</td>
<td>read bytes</td>
</tr>
<tr>
<td>gid</td>
<td>requester's group id</td>
</tr>
<tr>
<td>uid</td>
<td>requester's user id</td>
</tr>
<tr>
<td>size</td>
<td>read bytes</td>
</tr>
<tr>
<td>vlen</td>
<td>read blocks</td>
</tr>
<tr>
<td>offset</td>
<td>the offset of file</td>
</tr>
</tbody>
</table>
probe::nfsd.read

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

Synopsis

nfsd.read

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>read bytes</td>
</tr>
<tr>
<td>fh</td>
<td>file handle (the first part is the length of the file handle)</td>
</tr>
<tr>
<td>client_ip</td>
<td>the ip address of client</td>
</tr>
<tr>
<td>vec</td>
<td>struct kvec, includes buf address in kernel address and length of each buffer</td>
</tr>
<tr>
<td>offset</td>
<td>the offset of file</td>
</tr>
<tr>
<td>vlen</td>
<td>read blocks</td>
</tr>
<tr>
<td>size</td>
<td>read bytes</td>
</tr>
<tr>
<td>file</td>
<td>argument file, indicates if the file has been opened.</td>
</tr>
</tbody>
</table>
probe::nfsd.rename

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

Synopsis

nfsd.rename

Values

tlen length of new file name
fh file handler of old path
filename old file name
tfh file handler of new path
client_ip the ip address of client
tname new file name
flen length of old file name
probe::nfsd.unlink

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

Synopsis

nfsd.unlink

Values

- `filelen` the length of file name
- `fh` file handle (the first part is the length of the file handle)
- `filename` file name
- `type` file type (file or dir)
- `client_ip` the ip address of client
probe::nfsd.write

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

Synopsis

nfsd.write

Values

- **count**: read bytes
- **fh**: file handle (the first part is the length of the file handle)
- **client_ip**: the ip address of client
- **vec**: struct kvec, includes buf address in kernel address and length of each buffer
- **offset**: the offset of file
- **vlen**: read blocks
- **size**: read bytes
- **file**: argument file, indicates if the file has been opened.
Chapter 32. Speculation

This family of functions provides the ability to speculative record information and then at a later point in the SystemTap script either commit the information or discard it.
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

id	buffer id to store the information in
output	string to write out when commit occurs

Description

Add a string to the speculative 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

name The name of the array.
description Array description. An empty string can be used.

Description

This function adds a array, setting up everything needed. Arrays contain other metrics, added with json_add_array_numeric_metric or json_add_array_string_metric.
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_description:string,metric_units:string)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>array_name</td>
<td>The name of the array the numeric metric should be added to.</td>
</tr>
<tr>
<td>metric_name</td>
<td>The name of the numeric metric.</td>
</tr>
<tr>
<td>metric_description</td>
<td>Metric description. An empty string can be used.</td>
</tr>
<tr>
<td>metric_units</td>
<td>Metic units. An empty string can be used.</td>
</tr>
</tbody>
</table>

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

\[ \text{json\_add\_array\_string\_metric}: \long\langle \text{array\_name}: \text{string}, \text{metric\_name}: \text{string}, \text{metric\_description}: \text{string} \rangle \]

Arguments

- **array\_name**: The name of the array the string metric should be added to.
- **metric\_name**: The name of the string metric.
- **metric\_description**: Metric description. An empty string can be used.

Description

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

- name: The name of the numeric metric.
- description: Metric description. An empty string can be used.
- units: Metic units. An empty string can be used.

Description

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

function::json_add_string_metric — Add a string metric

Synopsis

\texttt{json\_add\_string\_metric:long(name:string,description:string)}

Arguments

name

The name of the string metric.

description

Metric description. An empty string can be used.

Description

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

function::json_set_prefix — Set the metric prefix.

**Synopsis**

```plaintext
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

array_name The name of the array.
array_index The array index (as a string) indicating where to store the numeric value.
metric_name The name of the numeric metric.
value The numeric value to output.

Description

The json_output_array_numeric_value macro is designed to be called from the 'json_data' probe in the user's script to output a metric's numeric value that is in an array. This metric should have been added with json_add_array_numeric_metric.
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

array_name The name of the array.
array_index The array index (as a string) indicating where to store the string value.
metric_name The name of the string metric.
value The string value to output.

Description

The json_output_array_string_value macro is designed to be called from the 'json_data' probe in the user's script to output a metric's string value that is in an array. This metric should have been added with json_add_array_string_metric.
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).
Chapter 35. syscalls

Following is an overview of available syscall probes and convenience variables they offer. By default, each syscall probe has name and argstr convenience variables, which are not included in the overview in order to keep it short. Non dwarf-based nd_syscall probes are supposed to have the same convenience variables.

Table 35.1. Syscalls list

<table>
<thead>
<tr>
<th>syscall</th>
<th>params</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept</td>
<td>addr_uaddr, addrlen_uaddr, sockfd</td>
</tr>
<tr>
<td>accept4</td>
<td>addr_uaddr, addrlen_uaddr, flags, flags_str, sockfd</td>
</tr>
<tr>
<td>access</td>
<td>mode, mode_str, pathname</td>
</tr>
<tr>
<td>acct</td>
<td>filename</td>
</tr>
<tr>
<td>add_key</td>
<td>description_uaddr, payload_uaddr, plen, ringid, type_uaddr</td>
</tr>
<tr>
<td>adjtimex</td>
<td>buf_str, buf_uaddr</td>
</tr>
<tr>
<td>alarm</td>
<td>seconds</td>
</tr>
<tr>
<td>arch_prctl</td>
<td>addr, code, code_str</td>
</tr>
<tr>
<td>bdflush</td>
<td>data, data_str, func</td>
</tr>
<tr>
<td>bind</td>
<td>addrlen, my_addr_uaddr, sockfd, uaddr_af, uaddr_ip, uaddr_ip_port,</td>
</tr>
<tr>
<td></td>
<td>uaddr_ipv6_flowinfo, uaddr_ipv6_scope_id</td>
</tr>
<tr>
<td>bpf</td>
<td>attr_uaddr, cmd, cmd_str, size</td>
</tr>
<tr>
<td>brk</td>
<td>brk</td>
</tr>
<tr>
<td>capget</td>
<td>data_uaddr, header_uaddr</td>
</tr>
<tr>
<td>capset</td>
<td>data_uaddr, header_uaddr</td>
</tr>
<tr>
<td>chdir</td>
<td>path</td>
</tr>
<tr>
<td>chmod</td>
<td>mode, path</td>
</tr>
<tr>
<td>chown</td>
<td>group, owner, path</td>
</tr>
<tr>
<td>chown16</td>
<td>group, owner, path</td>
</tr>
<tr>
<td>chroot</td>
<td>path</td>
</tr>
<tr>
<td>clock_adjtime</td>
<td>clk_id, clk_id_str, tx_uaddr</td>
</tr>
<tr>
<td>clock_getres</td>
<td>clk_id, clk_id_str, res_uaddr</td>
</tr>
<tr>
<td>clock_gettime</td>
<td>clk_id, clk_id_str, tp_uaddr</td>
</tr>
<tr>
<td>clock_nanosleep</td>
<td>clk_id, clk_id_str, flags, flags_str, rem_uaddr, req_uaddr</td>
</tr>
<tr>
<td>clock_settime</td>
<td>clk_id, clk_id_str, tp_uaddr</td>
</tr>
<tr>
<td>clone</td>
<td>child_tid_uaddr, clone_flags, parent_tid_uaddr, stack_start</td>
</tr>
<tr>
<td>close</td>
<td>fd</td>
</tr>
<tr>
<td>compat_adjtimex</td>
<td>buf_str, buf_uaddr</td>
</tr>
<tr>
<td>compat_clock_nanosleep</td>
<td>clk_id, clk_id_str, flags, flags_str, rem_uaddr, req_uaddr</td>
</tr>
<tr>
<td>compat_execve</td>
<td>args, env_str, filename</td>
</tr>
<tr>
<td>compat_execveat</td>
<td>args, dirfd, dirfd_str, env_str, filename, flags, flags_str</td>
</tr>
<tr>
<td>compat_fadvise64</td>
<td>advice, fd, len, offset</td>
</tr>
<tr>
<td>syscall</td>
<td>params</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>compat_fadvise64_64</td>
<td>advice, fd, len, offset</td>
</tr>
<tr>
<td>compat_fallocate</td>
<td>fd, len, mode, offset</td>
</tr>
<tr>
<td>compat_ftruncate64</td>
<td>fd, length</td>
</tr>
<tr>
<td>compat_futex</td>
<td>futex_uaddr, op, uaddr2_uaddr, utime_uaddr, val3, val</td>
</tr>
<tr>
<td>compat_futimesat</td>
<td>dirfd, dirfd_str, filename, filename_uaddr, tvp_uaddr</td>
</tr>
<tr>
<td>compat_getitimer</td>
<td>value_uaddr, which</td>
</tr>
<tr>
<td>compat_lookup_dcookie</td>
<td>buffer_uaddr, cookie, len</td>
</tr>
<tr>
<td>compat_nanosleep</td>
<td>rem_uaddr, req_uaddr</td>
</tr>
<tr>
<td>compat_ppoll</td>
<td></td>
</tr>
<tr>
<td>compat_pselect6</td>
<td>count, fd, offset</td>
</tr>
<tr>
<td>compat_readahead</td>
<td></td>
</tr>
<tr>
<td>compat_recvmmsg</td>
<td>flags, flags_str, mmsg_uaddr, s, timeout_uaddr, vlen</td>
</tr>
<tr>
<td>compat_rt_sigprocmask</td>
<td>how, how_str, oldset_uaddr, set_uaddr</td>
</tr>
<tr>
<td>compat_select</td>
<td>exceptfds_uaddr, n, readfds_uaddr, timeout_uaddr, writefds_uaddr</td>
</tr>
<tr>
<td>compat_setitimer</td>
<td>ovalue_uaddr, value_uaddr, which</td>
</tr>
<tr>
<td>compat_signalfd</td>
<td>flags</td>
</tr>
<tr>
<td>compat_sys_msgctl</td>
<td>buf_uaddr, cmd, cmd_str, msqid</td>
</tr>
<tr>
<td>compat_sys_msgrcv</td>
<td>msgflg, msgflg_str, msgp_uaddr, msgsz, msgtyp, msqid</td>
</tr>
<tr>
<td>compat_sys_msgsnd</td>
<td>msgflg, msgflg_str, msgp_uaddr, msgsz, msqid</td>
</tr>
<tr>
<td>compat_sys_recvmsg</td>
<td>flags, flags_str, msg_uaddr, s</td>
</tr>
<tr>
<td>compat_sys_semctl</td>
<td>arg, cmd, cmdstr, semid, semnum</td>
</tr>
<tr>
<td>compat_sys_semtimedop</td>
<td>nsops, semid, sops_uaddr, timeout_uaddr</td>
</tr>
<tr>
<td>compat_sys_sendmsg</td>
<td>flags, flags_str, msg_uaddr, s</td>
</tr>
<tr>
<td>compat_sys_shmat</td>
<td>shmaddr_uaddr, shflg, shmid</td>
</tr>
<tr>
<td>compat_sys_shmctl</td>
<td>buf_uaddr, cmd, cmd_str, shmid</td>
</tr>
<tr>
<td>compat_sys_utimes</td>
<td>filename</td>
</tr>
<tr>
<td>compat_truncate64</td>
<td>length, path, path_uaddr</td>
</tr>
<tr>
<td>compat_utime</td>
<td>actime, buf_uaddr, filename, filename_uaddr, modtime</td>
</tr>
<tr>
<td>compat_utimensat</td>
<td>dfd, dfd_str, filename, filename_uaddr, flags, flags_str, tsp_uaddr</td>
</tr>
<tr>
<td>compat_vmsplice</td>
<td>fd, flags, flags_str, iov, nr_segs</td>
</tr>
<tr>
<td>connect</td>
<td>addrlen, serv_addrlen_uaddr, sockfd, uaddr_af, uaddr_ip, uaddr_ip_port, uaddr_ipv6_flowinfo, uaddr_ipv6_scope_id</td>
</tr>
<tr>
<td>creat</td>
<td>mode, pathname</td>
</tr>
<tr>
<td>delete_module</td>
<td>flags, flags_str, name_user</td>
</tr>
<tr>
<td>dup</td>
<td>oldfd</td>
</tr>
<tr>
<td>dup2</td>
<td>flags, newfd, oldfd</td>
</tr>
<tr>
<td>dup3</td>
<td>flags, newfd, oldfd</td>
</tr>
<tr>
<td>epoll_create</td>
<td>flags, size</td>
</tr>
<tr>
<td>syscall</td>
<td>params</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>epoll_ctl</td>
<td>epfd, event_uaddr, fd, op, op_str</td>
</tr>
<tr>
<td>epoll_pwait</td>
<td>epfd, events_uaddr, maxevents, sigmask_uaddr, sigsetsize, timeout</td>
</tr>
<tr>
<td>epoll_wait</td>
<td>epfd, events_uaddr, maxevents, timeout</td>
</tr>
<tr>
<td>eventfd</td>
<td>count, flags, flags_str</td>
</tr>
<tr>
<td>execve</td>
<td>args, env_str, filename</td>
</tr>
<tr>
<td>execveat</td>
<td>args, dirfd, dirfd_str, env_str, filename, flags, flags_str</td>
</tr>
<tr>
<td>exit</td>
<td>status</td>
</tr>
<tr>
<td>exit_group</td>
<td>status</td>
</tr>
<tr>
<td>faccessat</td>
<td>dirfd, dirfd_str, mode, mode_str, pathname</td>
</tr>
<tr>
<td>fadvise64</td>
<td>advice, fd, len, offset</td>
</tr>
<tr>
<td>fadvise64_64</td>
<td>advice, fd, len, offset</td>
</tr>
<tr>
<td>fallocate</td>
<td>fd, len, mode, mode_str, offset</td>
</tr>
<tr>
<td>fanotify_init</td>
<td>event_f_flags, event_f_flags_str, flags, flags_str</td>
</tr>
<tr>
<td>fanotify_mark</td>
<td>dirfd, dirfd_str, fanotify_fd, flags, flags_str, mask, mask_str, pathname</td>
</tr>
<tr>
<td>fchdir</td>
<td>fd</td>
</tr>
<tr>
<td>chmod</td>
<td>fildes, mode</td>
</tr>
<tr>
<td>chmodat</td>
<td>dirfd, dirfd_str, mode, pathname</td>
</tr>
<tr>
<td>fchown</td>
<td>fd, group, owner</td>
</tr>
<tr>
<td>fchown16</td>
<td>fd, group, owner</td>
</tr>
<tr>
<td>fchownat</td>
<td>dirfd, dirfd_str, flags, flags_str, group, owner, pathname</td>
</tr>
<tr>
<td>fcntl</td>
<td>arg, cmd, cmd_str, fd</td>
</tr>
<tr>
<td>fdatasync</td>
<td>fd</td>
</tr>
<tr>
<td>fgetxattr</td>
<td>filedes, name_str, size, value_uaddr</td>
</tr>
<tr>
<td>finit_module</td>
<td>fd, flags, flags_str, uargs</td>
</tr>
<tr>
<td>flistxattr</td>
<td>filedes, list_uaddr, size</td>
</tr>
<tr>
<td>flock</td>
<td>fd, operation</td>
</tr>
<tr>
<td>fork</td>
<td></td>
</tr>
<tr>
<td>fremovexattr</td>
<td>filedes, name_str, name_uaddr</td>
</tr>
<tr>
<td>fsetxattr</td>
<td>filedes, flags, flags_str, name_str, name_uaddr, size, value_str, value_uaddr</td>
</tr>
<tr>
<td>fstat</td>
<td>buf_uaddr, filedes</td>
</tr>
<tr>
<td>fstatat</td>
<td>buf_uaddr, dirfd, dirfd_str, flags, flags_str, path</td>
</tr>
<tr>
<td>fstatfs</td>
<td>buf_uaddr, fd</td>
</tr>
<tr>
<td>fstatfs64</td>
<td>buf_uaddr, fd, sz</td>
</tr>
<tr>
<td>fsync</td>
<td>fd</td>
</tr>
<tr>
<td>ftruncate</td>
<td>fd, length</td>
</tr>
<tr>
<td>futex</td>
<td>futex_uaddr, op, uaddr2_uaddr, utime_uaddr, val3, val</td>
</tr>
<tr>
<td>futimesat</td>
<td>dirfd, dirfd_str, filename, filename_uaddr, tvp_uaddr</td>
</tr>
<tr>
<td>get_mempolicy</td>
<td>addr, flags, flags_str, maxnode, nmask_uaddr, policy_uaddr</td>
</tr>
<tr>
<td>syscall</td>
<td>params</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>get_robust_list</td>
<td>len_uaddr, list_head_uaddr, pid</td>
</tr>
<tr>
<td>get_thread_area</td>
<td>u_info_uaddr</td>
</tr>
<tr>
<td>getcpu</td>
<td>cpu_uaddr, node_uaddr, tcache_uaddr</td>
</tr>
<tr>
<td>getcwd</td>
<td>buf_uaddr, size</td>
</tr>
<tr>
<td>getdents</td>
<td>count, dirp_uaddr, fd</td>
</tr>
<tr>
<td>getegid</td>
<td></td>
</tr>
<tr>
<td>geteuid</td>
<td></td>
</tr>
<tr>
<td>getgid</td>
<td></td>
</tr>
<tr>
<td>getgroups</td>
<td>list_uaddr, size</td>
</tr>
<tr>
<td>gethostname</td>
<td>len, name_uaddr</td>
</tr>
<tr>
<td>getitimer</td>
<td>value_uaddr, which</td>
</tr>
<tr>
<td>getpeername</td>
<td>name_uaddr, namelen_uaddr, s</td>
</tr>
<tr>
<td>getpgid</td>
<td>pid</td>
</tr>
<tr>
<td>getpgrp</td>
<td></td>
</tr>
<tr>
<td>getpid</td>
<td></td>
</tr>
<tr>
<td>getppid</td>
<td></td>
</tr>
<tr>
<td>getpriority</td>
<td>which, who</td>
</tr>
<tr>
<td>getrandom</td>
<td>buf, count, flags, flags_str</td>
</tr>
<tr>
<td>getresgid</td>
<td>egid_uaddr, rgid_uaddr, sgid_uaddr</td>
</tr>
<tr>
<td>getresuid</td>
<td>euid_uaddr, ruid_uaddr, suid_uaddr</td>
</tr>
<tr>
<td>getrlimit</td>
<td>resource, rlim_uaddr</td>
</tr>
<tr>
<td>getrusage</td>
<td>usage_uaddr, who, who_str</td>
</tr>
<tr>
<td>getsid</td>
<td>pid</td>
</tr>
<tr>
<td>getsockname</td>
<td>name_uaddr, namelen_uaddr, s</td>
</tr>
<tr>
<td>getsockopt</td>
<td>fd, level, level_str, optlen_uaddr, optname, optname_str, optval_uaddr</td>
</tr>
<tr>
<td>gettid</td>
<td></td>
</tr>
<tr>
<td>gettimeofday</td>
<td>tv_uaddr, tz_uaddr</td>
</tr>
<tr>
<td>getuid</td>
<td></td>
</tr>
<tr>
<td>getxattr</td>
<td>name_str, path, size, value_uaddr</td>
</tr>
<tr>
<td>init_module</td>
<td>len, uargs, umod_uaddr</td>
</tr>
<tr>
<td>inotify_add_watch</td>
<td>fd, mask, path, path_uaddr</td>
</tr>
<tr>
<td>inotify_init</td>
<td>flags</td>
</tr>
<tr>
<td>inotify_rm_watch</td>
<td>fd, wd</td>
</tr>
<tr>
<td>io_cancel</td>
<td>ctx_id, iocb_uaddr, result_uaddr</td>
</tr>
<tr>
<td>io_destroy</td>
<td>ctx</td>
</tr>
<tr>
<td>io_getevents</td>
<td>ctx_id, events_uaddr, min_nr, nr, timeout_uaddr, timestr</td>
</tr>
<tr>
<td>io_setup</td>
<td>ctxp_uaddr, maxevents</td>
</tr>
<tr>
<td>io_submit</td>
<td>ctx_id, iocbpp_uaddr, nr</td>
</tr>
</tbody>
</table>
### syscall | params
---|---
ioctl | argp, fd, request
>ioperm | from, num, turn_on
>ioctl | argp, fd, request
>iopl | level
>ioctl | argp, fd, request
>ioprio_get | which, which_str, who
>ioprio_set | ioprio, ioprio_str, which, which_str, who
>kcmp | idx1, idx2, pid1, pid2, type, type_str
>kexec_file_load | cmdline, cmdline_len, flags, flags_str, initrd_fd, kernel_fd
>kexec_load | entry, flags, flags_str, nr_segments, segments_uaddr
>keyctl | arg2, arg3, arg4, arg5, option
>kill | pid, sig
>lchown | group, owner, path
>lchown16 | group, owner, path
>lgetxattr | name_str, path, size, value_uaddr
>link | newpath, oldpath
>linkat | flags, flags_str, newdirfd, newdirfd_str, newpath, olddirfd, olddirfd_str, oldpath
>listen | backlog, sockfd
>listxattr | list_uaddr, path, path_uaddr, size
>llistxattr | list_uaddr, path, path_uaddr, size
>lseek | fd, offset_high, offset_low, result_uaddr, whence, whence_str
>lookup_dcookie | buffer_uaddr, cookie, len
>lremovexattr | name_str, name_uaddr, path, path_uaddr
>lseek | fildes, offset, whence, whence_str
>lsetxattr | flags, flags_str, name_str, name_uaddr, path, path_uaddr, size, value_str, value_uaddr
>lstat | buf_uaddr, path
>advise | advice, advice_str, length, start
>mbind | flags, flags_str, len, maxnode, mode, mode_str, nmask_uaddr, start
>memfd_create | flags, flags_str, uname
>migrate_pages | maxnode, new_nodes, old_nodes, pid
>mincore | length, start, vec_uaddr
>mkdir | mode, pathname, pathname_uaddr
>mkdirat | dirfd, dirfd_str, mode, pathname
>mkfs | dev, mode, pathname
>mkfsat | dev, dirfd, dirfd_str, mode, mode_str, pathname
>mlock | addr, len
>mlockall | flags
>mmap2 | fd, flags, length, pgoffset, prot, start
<table>
<thead>
<tr>
<th>syscall</th>
<th>params</th>
</tr>
</thead>
<tbody>
<tr>
<td>modify_ldt</td>
<td>bytecount, func, ptr_uaddr</td>
</tr>
<tr>
<td>mount</td>
<td>data, filesystemtype, mountflags, mountflags_str, source, target</td>
</tr>
<tr>
<td>move_pages</td>
<td>flags, flags_str, nodes, nr_pages, pages, pid, status</td>
</tr>
<tr>
<td>mprotect</td>
<td>addr, len, prot, prot_str</td>
</tr>
<tr>
<td>mq_getsetattr</td>
<td>mqdes, u_mqstat_uaddr, u_omqstat_uaddr</td>
</tr>
<tr>
<td>mq_notify</td>
<td>mqdes, notification_uaddr</td>
</tr>
<tr>
<td>mq_open</td>
<td>filename, mode, name_uaddr, oflag, u_attr_uaddr</td>
</tr>
<tr>
<td>mq_timedreceive</td>
<td>abs_timeout_uaddr, mqdes, msg_len, msg_prio_uaddr, msg_ptr_uaddr</td>
</tr>
<tr>
<td>mq_timedsend</td>
<td>abs_timeout_uaddr, mqdes, msg_len, msg_prio, msg_ptr_uaddr</td>
</tr>
<tr>
<td>mq_unlink</td>
<td>u_name, u_name_uaddr</td>
</tr>
<tr>
<td>mremap</td>
<td>flags, new_address, new_size, old_address, old_size</td>
</tr>
<tr>
<td>msgctl</td>
<td>buf_uaddr, cmd, cmd_str, msqid</td>
</tr>
<tr>
<td>msgget</td>
<td>key, key_str, msgflg, msgflg_str</td>
</tr>
<tr>
<td>msgrcv</td>
<td>msgflg, msgflg_str, msgp_uaddr, msgsz, msgtyp, msqid</td>
</tr>
<tr>
<td>msgsnd</td>
<td>msgflg, msgflg_str, msgp_uaddr, msgsz, msqid</td>
</tr>
<tr>
<td>msync</td>
<td>flags, length, start</td>
</tr>
<tr>
<td>munlock</td>
<td>addr, len</td>
</tr>
<tr>
<td>munlockall</td>
<td></td>
</tr>
<tr>
<td>munmap</td>
<td>length, start</td>
</tr>
<tr>
<td>name_to_handle_at</td>
<td>dfd, dfd_str, flags, flags_str, handle_uaddr, mnt_id_uaddr, pathname</td>
</tr>
<tr>
<td>nanosleep</td>
<td>rem_uaddr, req_uaddr</td>
</tr>
<tr>
<td>ni_syscall</td>
<td></td>
</tr>
<tr>
<td>nice</td>
<td>inc</td>
</tr>
<tr>
<td>open</td>
<td>filename, flags, mode</td>
</tr>
<tr>
<td>open_by_handle_at</td>
<td>flags, flags_str, handle_uaddr, mountdfd, mountdfd_str</td>
</tr>
<tr>
<td>openat</td>
<td>dfd, dfd_str, filename, flags, mode</td>
</tr>
<tr>
<td>pause</td>
<td></td>
</tr>
<tr>
<td>perf_event_open</td>
<td>attr_uaddr, cpu, flags, flags_str, group_fd, pid</td>
</tr>
<tr>
<td>personality</td>
<td>persona</td>
</tr>
<tr>
<td>pipe</td>
<td>fildes_uaddr, flag_str, flags, pipe0, pipe1</td>
</tr>
<tr>
<td>pivot_root</td>
<td>new_root_str, old_root_str</td>
</tr>
<tr>
<td>poll</td>
<td>nfds, timeout, ufds_uaddr</td>
</tr>
<tr>
<td>ppoll</td>
<td></td>
</tr>
<tr>
<td>prctl</td>
<td>arg2, arg3, arg4, arg5, option</td>
</tr>
<tr>
<td>pread</td>
<td>buf_uaddr, count, fd, offset</td>
</tr>
<tr>
<td>preadv</td>
<td>count, fd, offset, vector_uaddr</td>
</tr>
<tr>
<td>prlimit64</td>
<td>new_rlim_uaddr, old_rlim_uaddr, pid, resource</td>
</tr>
<tr>
<td>process_vm_readv</td>
<td>flags, liovent, local_iov_uaddr, pid, remote_iov_uaddr, riovcnt</td>
</tr>
<tr>
<td>syscall</td>
<td>params</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>process_vm_writev</td>
<td>flags, liovcnt, local_iov_uaddr, pid, remote_iov_uaddr, riovcnt</td>
</tr>
<tr>
<td>pselect6</td>
<td></td>
</tr>
<tr>
<td>ptrace</td>
<td>addr, data, pid, request</td>
</tr>
<tr>
<td>pwrite</td>
<td>buf_uaddr, count, fd, offset</td>
</tr>
<tr>
<td>pwritev</td>
<td>count, fd, offset, vector_uaddr</td>
</tr>
<tr>
<td>quotactl</td>
<td>addr_uaddr, cmd, cmd_str, id, special, special_str</td>
</tr>
<tr>
<td>read</td>
<td>buf_uaddr, count, fd</td>
</tr>
<tr>
<td>readahead</td>
<td>count, fd, offset</td>
</tr>
<tr>
<td>readdir</td>
<td>count, dirent, fd</td>
</tr>
<tr>
<td>readlink</td>
<td>buf_uaddr, bufsiz, path</td>
</tr>
<tr>
<td>readdirat</td>
<td>buf_uaddr, bufsiz, dfd, dfd_str, path</td>
</tr>
<tr>
<td>readv</td>
<td>count, fd, vector_uaddr</td>
</tr>
<tr>
<td>reboot</td>
<td>arg_uaddr, flag, flag_str, magic2, magic2_str, magic, magic_str</td>
</tr>
<tr>
<td>recv</td>
<td>buf_uaddr, flags, flags_str, len, s</td>
</tr>
<tr>
<td>recvfrom</td>
<td>addr_uaddr, addrlen_uaddr, buf_uaddr, flags, flags_str, len, s</td>
</tr>
<tr>
<td>recvmsg</td>
<td>flags, flags_str, mmsg_uaddr, s, timeout_uaddr, vlen</td>
</tr>
<tr>
<td>remap_file_pages</td>
<td>flags, pgoff, prot, size, start</td>
</tr>
<tr>
<td>removexattr</td>
<td>name_str, path</td>
</tr>
<tr>
<td>rename</td>
<td>newpath, oldpath</td>
</tr>
<tr>
<td>renameat</td>
<td>newdfd, newdfd_str, newname, newname_str, olddfd, olddfd_str, oldname, oldname_str</td>
</tr>
<tr>
<td>renameat2</td>
<td>flags, flags_str, newdfd, newdfd_str, newname, newname_str, olddfd, olddfd_str, oldname, oldname_str</td>
</tr>
<tr>
<td>request_key</td>
<td>callout_info_uaddr, description_uaddr, destringid, type_uaddr</td>
</tr>
<tr>
<td>restart_syscall</td>
<td></td>
</tr>
<tr>
<td>rmdir</td>
<td>pathname</td>
</tr>
<tr>
<td>rt_sigaction</td>
<td>act_uaddr, oact_uaddr, sig, sigsetsize</td>
</tr>
<tr>
<td>rt_sigaction32</td>
<td>act_uaddr, oact_uaddr, sig, sigsetsize</td>
</tr>
<tr>
<td>rt_sigpending</td>
<td>set_uaddr, sigsetsize</td>
</tr>
<tr>
<td>rt_sigprocmask</td>
<td>how, how_str, oldset_uaddr, set_uaddr</td>
</tr>
<tr>
<td>rt_sigqueueinfo</td>
<td>pid, sig, uinfo_uaddr</td>
</tr>
<tr>
<td>rt_sigreturn</td>
<td></td>
</tr>
<tr>
<td>rt_sigsuspend</td>
<td>set_uaddr, sigsetsize</td>
</tr>
<tr>
<td>rt_sigtimedwait</td>
<td>sigsetsize, uinfo_uaddr, uthese_uaddr, uts_uaddr</td>
</tr>
<tr>
<td>rt_tgsigqueueinfo</td>
<td>sig, tgid, tid, uinfo_uaddr</td>
</tr>
<tr>
<td>sched_get_priority_max</td>
<td>policy</td>
</tr>
<tr>
<td>sched_get_priority_min</td>
<td>policy</td>
</tr>
<tr>
<td>syscall</td>
<td>params</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>sched_getaffinity</td>
<td>len, mask_uaddr, pid</td>
</tr>
<tr>
<td>sched_getattr</td>
<td>flags, pid, sched_attr_str, sched_attr_uaddr, size</td>
</tr>
<tr>
<td>sched_getparam</td>
<td>p_uaddr, pid</td>
</tr>
<tr>
<td>sched_getscheduler</td>
<td>pid</td>
</tr>
<tr>
<td>sched_rr_get_interval</td>
<td>pid, tp_uaddr</td>
</tr>
<tr>
<td>sched_setaffinity</td>
<td>len, mask_uaddr, pid</td>
</tr>
<tr>
<td>sched_setattr</td>
<td>flags, pid, sched_attr_str, sched_attr_uaddr</td>
</tr>
<tr>
<td>sched_setparam</td>
<td>p_uaddr, pid</td>
</tr>
<tr>
<td>sched_setscheduler</td>
<td>p_uaddr, pid, policy, policy_str</td>
</tr>
<tr>
<td>sched_yield</td>
<td></td>
</tr>
<tr>
<td>seccomp</td>
<td>flags, flags_str, op, op_str, uargs, uargs_uaddr</td>
</tr>
<tr>
<td>select</td>
<td>exceptfds_uaddr, n, readfds_uaddr, timeout_uaddr, writefds_uaddr</td>
</tr>
<tr>
<td>semctl</td>
<td>arg, cmd, cmdstr, semid, semnum</td>
</tr>
<tr>
<td>semget</td>
<td>key, key_str, nsems, semflg, semflg_str</td>
</tr>
<tr>
<td>semop</td>
<td>nsops, semid, sops_uaddr</td>
</tr>
<tr>
<td>semtimedop</td>
<td>nsops, semid, sops_uaddr, timeout_uaddr</td>
</tr>
<tr>
<td>send</td>
<td>buf_uaddr, flags, flags_str, len, s</td>
</tr>
<tr>
<td>sendfile</td>
<td>count, in_fd, offset_uaddr, out_fd</td>
</tr>
<tr>
<td>sendmmsg</td>
<td>flags, flags_str, mmsg_uaddr, s, vlen</td>
</tr>
<tr>
<td>sendmsg</td>
<td>flags, flags_str, msg_uaddr, s</td>
</tr>
<tr>
<td>sendto</td>
<td>buf_uaddr, flags, flags_str, len, s, to_uaddr, tolen</td>
</tr>
<tr>
<td>set_mempolicy</td>
<td>maxnode, mode, mode_str, nmask_uaddr</td>
</tr>
<tr>
<td>set_robust_list</td>
<td>len, list_head_uaddr</td>
</tr>
<tr>
<td>set_thread_area</td>
<td>u_info_uaddr</td>
</tr>
<tr>
<td>set_tid_address</td>
<td>tidptr_uaddr</td>
</tr>
<tr>
<td>setdomainname</td>
<td>domainname_str, domainname_uaddr, len</td>
</tr>
<tr>
<td>setfsgid</td>
<td>fsgid</td>
</tr>
<tr>
<td>setfsuid</td>
<td>fsuid</td>
</tr>
<tr>
<td>setgid</td>
<td>gid</td>
</tr>
<tr>
<td>setgroups</td>
<td>list_uaddr, size</td>
</tr>
<tr>
<td>sethostname</td>
<td>hostname_uaddr, len, name_str</td>
</tr>
<tr>
<td>setitimer</td>
<td>ovalue_uaddr, value_uaddr, which</td>
</tr>
<tr>
<td>sets</td>
<td>fd, nstype</td>
</tr>
<tr>
<td>setpgid</td>
<td>pgid, pid</td>
</tr>
<tr>
<td>setpriority</td>
<td>prio, which, which_str, who</td>
</tr>
<tr>
<td>setregid</td>
<td>egid, rgid</td>
</tr>
<tr>
<td>setregid16</td>
<td>egid, rgid</td>
</tr>
<tr>
<td>setresgid</td>
<td>egid, rgid, sgid</td>
</tr>
<tr>
<td>syscall</td>
<td>params</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>setresgid16</td>
<td>egid, rgid, sgid</td>
</tr>
<tr>
<td>setresuid</td>
<td>euid, ruid, suid</td>
</tr>
<tr>
<td>setresuid16</td>
<td>euid, ruid, suid</td>
</tr>
<tr>
<td>setreuid</td>
<td>euid, ruid</td>
</tr>
<tr>
<td>setreuid16</td>
<td>euid, ruid</td>
</tr>
<tr>
<td>setrlimit</td>
<td>resource, rlim_uaddr</td>
</tr>
<tr>
<td>setsid</td>
<td></td>
</tr>
<tr>
<td>setsockopt</td>
<td>fd, level, level_str, optlen, optname, optname_str, optval_uaddr</td>
</tr>
<tr>
<td>settimeofday</td>
<td>tv_uaddr, tz_uaddr</td>
</tr>
<tr>
<td>settimeofday32</td>
<td>tv_uaddr, tz_uaddr</td>
</tr>
<tr>
<td>setuid</td>
<td>uid</td>
</tr>
<tr>
<td>setxattr</td>
<td>flags, flags_str, name_str, name_uaddr, path, path_uaddr, size, value_str, value_uaddr</td>
</tr>
<tr>
<td>sgetmask</td>
<td></td>
</tr>
<tr>
<td>shmat</td>
<td>shmaddr_uaddr, shmfldg, shmid</td>
</tr>
<tr>
<td>shmatl</td>
<td>buf_uaddr, cmd, cmd_str, shmid</td>
</tr>
<tr>
<td>shmdt</td>
<td>shmaddr_uaddr</td>
</tr>
<tr>
<td>shmget</td>
<td>key, shmfldg, shmfldg_str, size</td>
</tr>
<tr>
<td>shutdown</td>
<td>how, how_str, s</td>
</tr>
<tr>
<td>sigaction32</td>
<td>act_uaddr, oact_uaddr, sig</td>
</tr>
<tr>
<td>sigaltstack</td>
<td>uoss_uaddr, uss_uaddr</td>
</tr>
<tr>
<td>signal</td>
<td>handler, sig</td>
</tr>
<tr>
<td>signalfd</td>
<td>flags</td>
</tr>
<tr>
<td>sigpending</td>
<td></td>
</tr>
<tr>
<td>sigprocmask</td>
<td>how, how_str, oldset_uaddr, set_uaddr</td>
</tr>
<tr>
<td>sigreturn</td>
<td></td>
</tr>
<tr>
<td>sigsuspend</td>
<td>mask</td>
</tr>
<tr>
<td>socket</td>
<td>family, protocol, type</td>
</tr>
<tr>
<td>socketpair</td>
<td>family, protocol, sv_uaddr, type</td>
</tr>
<tr>
<td>splice</td>
<td>fd_in, fd_out, flags, flags_str, len, off_in, off_out</td>
</tr>
<tr>
<td>ssetmask</td>
<td>newmask, newmask_str</td>
</tr>
<tr>
<td>stat</td>
<td>buf_uaddr, filename, filename_uaddr</td>
</tr>
<tr>
<td>statfs</td>
<td>buf_uaddr, path</td>
</tr>
<tr>
<td>statfs64</td>
<td>buf_uaddr, path</td>
</tr>
<tr>
<td>stime</td>
<td>t_uaddr</td>
</tr>
<tr>
<td>swapoff</td>
<td>path</td>
</tr>
<tr>
<td>swapon</td>
<td>path, swapflags, swapflags_str</td>
</tr>
<tr>
<td>symlink</td>
<td>newpath, oldpath</td>
</tr>
<tr>
<td>syscall</td>
<td>params</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>symlinkat</td>
<td>newdfd, newdfd_str, newname, newname_str, oldname, oldname_str</td>
</tr>
<tr>
<td>sync</td>
<td></td>
</tr>
<tr>
<td>sync_file_range</td>
<td>fd, flags, flags_str, nbytes, offset</td>
</tr>
<tr>
<td>syncfs</td>
<td></td>
</tr>
<tr>
<td>sysctl</td>
<td></td>
</tr>
<tr>
<td>sysfs</td>
<td>arg1, arg2, option</td>
</tr>
<tr>
<td>syslog</td>
<td></td>
</tr>
<tr>
<td>tee</td>
<td></td>
</tr>
<tr>
<td>tgkill</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td></td>
</tr>
<tr>
<td>timer_create</td>
<td>clockid, clockid_str, evp_uaddr, timerid_uaddr</td>
</tr>
<tr>
<td>timer_delete</td>
<td></td>
</tr>
<tr>
<td>timer_getoverrun</td>
<td></td>
</tr>
<tr>
<td>timer_gettime</td>
<td></td>
</tr>
<tr>
<td>timer_settime</td>
<td></td>
</tr>
<tr>
<td>timerfd_create</td>
<td></td>
</tr>
<tr>
<td>timerfd_gettime</td>
<td></td>
</tr>
<tr>
<td>timerfd_settime</td>
<td></td>
</tr>
<tr>
<td>times</td>
<td></td>
</tr>
<tr>
<td>tkill</td>
<td></td>
</tr>
<tr>
<td>truncate</td>
<td></td>
</tr>
<tr>
<td>umask</td>
<td></td>
</tr>
<tr>
<td>umount</td>
<td></td>
</tr>
<tr>
<td>uname</td>
<td></td>
</tr>
<tr>
<td>unlink</td>
<td></td>
</tr>
<tr>
<td>unlinkat</td>
<td></td>
</tr>
<tr>
<td>unshare</td>
<td></td>
</tr>
<tr>
<td>unstat</td>
<td></td>
</tr>
<tr>
<td>unstat32</td>
<td></td>
</tr>
<tr>
<td>utime</td>
<td></td>
</tr>
<tr>
<td>utimensat</td>
<td></td>
</tr>
<tr>
<td>utimes</td>
<td></td>
</tr>
<tr>
<td>vfork</td>
<td></td>
</tr>
<tr>
<td>vhangup</td>
<td></td>
</tr>
<tr>
<td>vmsplice</td>
<td></td>
</tr>
<tr>
<td>wait4</td>
<td></td>
</tr>
<tr>
<td>waitid</td>
<td></td>
</tr>
</tbody>
</table>

715
<table>
<thead>
<tr>
<th>syscall</th>
<th>params</th>
</tr>
</thead>
<tbody>
<tr>
<td>waitpid</td>
<td>options, options_str, pid, status_uaddr</td>
</tr>
<tr>
<td>write</td>
<td>buf_uaddr, count, fd</td>
</tr>
<tr>
<td>writev</td>
<td>count, fd, vector_uaddr</td>
</tr>
</tbody>
</table>