

SystemTap Tapset Reference Manual

SystemTap Tapset Reference Manual

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

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

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

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

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

Table of Contents

1. Introduction	1
Tapset Name Format	1
2. Context Functions	2
print_regs	3
execname	4
pid	5
tid	6
ppid	7
pexecname	8
gid	9
egid	10
uid	11
euid	12
cpu	13
pp	14
registers_valid	15
user_mode	16
is_return	17
target	18
stack_size	19
stack_used	20
stack_unused	21
print_stack	22
probefunc	23
probemod	24
print_backtrace	25
backtrace	26
caller	27
caller_addr	28
3. Timestamp Functions	29
get_cycles	30
4. Memory Tapset	31
vm_fault_contains	32
vm.pagefault	33
vm.pagefault.return	34
addr_to_node	35
vm.write_shared	36
vm.write_shared_copy	37
vm.mmap	38
vm.munmap	39
vm.brk	40
vm.oom_kill	41
5. IO Scheduler Tapset	42
ioscheduler.elv_next_request	43
ioscheduler.elv_next_request.return	44
ioscheduler.elv_add_request	45
ioscheduler.elv_completed_request	46
6. SCSI Tapset	47
scsi.ioentry	48
scsi.iodispatching	49
scsi.iodone	50
scsi.iocompleted	51
7. Networking Tapset	52
netdev.receive	53
netdev.transmit	54
tcp.sendmsg	55

tcp.sendmsg.return	56
tcp.recvmsg	57
tcp.recvmsg.return	58
tcp.disconnect	59
tcp.disconnect.return	60
tcp.setsockopt	61
tcp.setsockopt.return	62
udp.sendmsg	63
udp.sendmsg.return	64
udp.recvmsg	65
udp.recvmsg.return	66
udp.disconnect	67
udp.disconnect.return	68
ip_ntop	69
8. Socket Tapset	70
socket.send	71
socket.receive	72
socket.sendmsg	73
socket.sendmsg.return	74
socket.recvmsg	75
socket.recvmsg.return	76
socket.aio_write	77
socket.aio_write.return	78
socket.aio_read	79
socket.aio_read.return	80
socket.writev	81
socket.writev.return	82
socket.readv	83
socket.readv.return	84
socket.create	85
socket.create.return	86
socket.close	87
socket.close.return	88
sock_prot_num2str	89
sock_prot_str2num	90
sock_fam_num2str	91
sock_fam_str2num	92
sock_state_num2str	93
sock_state_str2num	94
9. Process Tapset	95
process.create	96
process.start	97
process.exec	98
process.exec_complete	99
process.exit	100
process.release	101
10. Signal Tapset	102
signal.send	103
signal.send.return	104
signal.checkperm	105
signal.checkperm.return	106
signal.wakeup	107
signal.check_ignored	108
signal.check_ignored.return	109
signal.force_segv	110
signal.force_segv.return	111
signal.syskill	112
signal.syskill.return	113

signal.sys_tkill	114
signal.systkill.return	115
signal.sys_tgkill	116
signal.sys_tgkill.return	117
signal.send_sig_queue	118
signal.send_sig_queue.return	119
signal.pending	120
signal.pending.return	121
signal.handle	122
signal.handle.return	123
signal.do_action	124
signal.do_action.return	125
signal.procmask	126
signal.flush	127

Chapter 1. Introduction

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

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

Tapset Name Format

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

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

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

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

Chapter 2. Context Functions

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

Name

`print_regs` — Print a register dump.

Synopsis

```
print_regs()
```

Arguments

None

Name

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

Synopsis

```
execname:string()
```

Arguments

None

Name

pid — Returns the ID of a target process.

Synopsis

```
pid:long()
```

Arguments

None

Name

`tid` — Returns the thread ID of a target process.

Synopsis

```
tid:long()
```

Arguments

None

Name

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

Synopsis

```
ppid:long()
```

Arguments

None

Name

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

Synopsis

```
pexecname:string()
```

Arguments

None

Name

gid — Returns the group ID of a target process.

Synopsis

```
gid:long()
```

Arguments

None

Name

`egid` — Returns the effective gid of a target process.

Synopsis

```
egid:long()
```

Arguments

None

Name

uid — Returns the user ID of a target process.

Synopsis

```
uid:long()
```

Arguments

None

Name

`eid` — Return the effective uid of a target process.

Synopsis

```
eid:long()
```

Arguments

None

Name

cpu — Returns the current cpu number.

Synopsis

```
cpu:long()
```

Arguments

None

Name

pp — Return the probe point associated with the currently running probe handler,

Synopsis

```
pp:string()
```

Arguments

None

Description

including alias and wildcard expansion effects

Context

The current probe point.

Name

`registers_valid` — Determines validity of `<command>register</command>` and `<command>u_register</command>` in current context.

Synopsis

```
registers_valid:long()
```

Arguments

None

Description

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

Name

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

Synopsis

```
user_mode:long()
```

Arguments

None

Description

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

Name

`is_return` — Determines if probe point is a return probe.

Synopsis

```
is_return:long()
```

Arguments

None

Description

Return 1 if the probe point is a return probe. *Deprecated.*

Name

target — Return the process ID of the target process.

Synopsis

```
target:long()
```

Arguments

None

Name

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

Synopsis

```
stack_size:long()
```

Arguments

None

Name

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

Synopsis

```
stack_used:long()
```

Arguments

None

Description

Determines how many bytes are currently used in the kernel stack.

Name

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

Synopsis

```
stack_unused:long()
```

Arguments

None

Description

Determines how many bytes are currently available in the kernel stack.

Name

`print_stack` — Print out stack from string.

Synopsis

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

Arguments

stk String with list of hexadecimal addresses. (FIXME)

Description

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

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

Name

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

Synopsis

```
probefunc:string()
```

Arguments

None

Name

probemod — Return the probe point's module name, if known.

Synopsis

```
probemod:string()
```

Arguments

None

Name

`print_backtrace` — Print stack back trace

Synopsis

```
print_backtrace()
```

Arguments

None

Description

Equivalent to `<command>print_stack(backtrace)</command>`, except that deeper stack nesting may be supported. Return nothing.

Name

backtrace — Hex backtrace of current stack

Synopsis

```
backtrace:string()
```

Arguments

None

Description

Return a string of hex addresses that are a backtrace of the stack. Output may be truncated as per maximum string length.

Name

caller — Return name and address of calling function

Synopsis

```
caller:string()
```

Arguments

None

Description

Return the address and name of the calling function. <emphasis>Works only for return probes at this time.</emphasis>

Name

caller_addr — Return caller address

Synopsis

```
caller_addr:long()
```

Arguments

None

Description

Return the address of the calling function. **Works only for return probes at this time.**

Chapter 3. Timestamp Functions

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

Name

`get_cycles` — Processor cycle count.

Synopsis

```
get_cycles:long()
```

Arguments

None

Description

Return the processor cycle counter value, or 0 if unavailable.

Chapter 4. Memory Tapset

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

Name

`vm_fault_contains` — Test return value for page fault reason

Synopsis

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

Arguments

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

Name

vm.pagefault — Records that a page fault occurred.

Synopsis

vm.pagefault

Values

write_access Indicates whether this was a write or read access; `<command>1</command>` indicates a write, while `<command>0</command>` indicates a read.

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

Context

The process which triggered the fault

Name

`vm.pagefault.return` — Indicates what type of fault occurred.

Synopsis

`vm.pagefault.return`

Values

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

Name

`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.

Name

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

Synopsis

```
vm.write_shared
```

Values

address The address of the shared write.

Context

The context is the process attempting the write.

Description

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

Name

`vm.write_shared_copy` — Page copy for shared page write.

Synopsis

`vm.write_shared_copy`

Values

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

address The address of the shared write.

Context

The process attempting the write.

Description

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

Name

vm.mmap — Fires when an `<command>mmap</command>` is requested.

Synopsis

`vm.mmap`

Values

length The length of the memory segment

address The requested address

Context

The process calling `<command>mmap</command>`.

Name

vm.munmap — Fires when an `<command>munmap</command>` is requested.

Synopsis

`vm.munmap`

Values

length The length of the memory segment

address The requested address

Context

The process calling `<command>munmap</command>`.

Name

vm.brk — Fires when a `<command>brk</command>` is requested (i.e. the heap will be resized).

Synopsis

`vm.brk`

Values

length The length of the memory segment

address The requested address

Context

The process calling `<command>brk</command>`.

Name

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

Synopsis

```
vm.oom_kill
```

Values

task The task being killed

Context

The process that tried to consume excessive memory, and thus triggered the OOM. <remark>(is this correct?)</remark>

Chapter 5. IO Scheduler Tapset

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

Name

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

Synopsis

`ioscheduler.elv_next_request`

Values

<code>elevator_name</code>	The type of I/O elevator currently enabled
----------------------------	--

Name

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

Synopsis

```
ioscheduler.elv_next_request.return
```

Values

<i>req_flags</i>	Request flags
<i>req</i>	Address of the request
<i>disk_major</i>	Disk major number of the request
<i>disk_minor</i>	Disk minor number of the request

Name

`ioscheduler.elv_add_request` — A request was added to the request queue

Synopsis

`ioscheduler.elv_add_request`

Values

<code>req_flags</code>	Request flags
<code>req</code>	Address of the request
<code>disk_major</code>	Disk major number of the request
<code>elevator_name</code>	The type of I/O elevator currently enabled
<code>disk_minor</code>	Disk minor number of the request

Name

`ioscheduler.elv_completed_request` — Fires when a request is completed

Synopsis

`ioscheduler.elv_completed_request`

Values

<code>req_flags</code>	Request flags
<code>req</code>	Address of the request
<code>disk_major</code>	Disk major number of the request
<code>elevator_name</code>	The type of I/O elevator currently enabled
<code>disk_minor</code>	Disk minor number of the request

Chapter 6. SCSI Tapset

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

Name

`scsi.ioentry` — Prepares a SCSI mid-layer request

Synopsis

```
scsi.ioentry
```

Values

<code>disk_major</code>	The major number of the disk (-1 if no information)
<code>device_state</code>	The current state of the device.
<code>disk_minor</code>	The minor number of the disk (-1 if no information)

Name

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

Synopsis

`scsi.iodispatching`

Values

<i>lun</i>	The lun number
<i>req_bufflen</i>	The request buffer length
<i>host_no</i>	The host number
<i>device_state</i>	The current state of the device.
<i>dev_id</i>	The scsi device id
<i>channel</i>	The channel number
<i>data_direction</i>	The <code>data_direction</code> specifies whether this command is from/to the device. 0 (DMA_BIDIRECTIONAL), 1 (DMA_TO_DEVICE), 2 (DMA_FROM_DEVICE), 3 (DMA_NONE)
<i>request_buffer</i>	The request buffer address

Name

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

Synopsis

`scsi.iodone`

Values

<i>lun</i>	The lun number
<i>host_no</i>	The host number
<i>device_state</i>	The current state of the device
<i>dev_id</i>	The scsi device id
<i>channel</i>	The channel number
<i>data_direction</i>	The <code>data_direction</code> specifies whether this command is from/to the device.

Name

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

Synopsis

`scsi.iocompleted`

Values

<i>lun</i>	The lun number
<i>host_no</i>	The host number
<i>device_state</i>	The current state of the device
<i>dev_id</i>	The scsi device id
<i>channel</i>	The channel number
<i>data_direction</i> device	The <code>data_direction</code> specifies whether this command is from/to the
<i>goodbytes</i>	The bytes completed.

Chapter 7. Networking Tapset

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

Name

netdev.receive — Data recieved from network device.

Synopsis

```
netdev.receive
```

Values

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

Name

netdev.transmit — Network device transmitting buffer

Synopsis

```
netdev.transmit
```

Values

<i>protocol</i>	The protocol of this packet.
<i>dev_name</i>	The name of the device. e.g: eth0, ath1.
<i>length</i>	The length of the transmit buffer.
<i>true_size</i>	The size of the the data to be transmitted.

Name

tcp.sendmsg — Sending a tcp message

Synopsis

`tcp.sendmsg`

Values

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

Context

The process which sends a tcp message

Name

tcp.sendmsg.return — Sending TCP message is done

Synopsis

```
tcp.sendmsg.return
```

Values

<i>name</i>	Name of this probe
<i>size</i>	Number of bytes sent or error code if an error occurred.

Context

The process which sends a tcp message

Name

tcp.recvmsg — Receiving TCP message

Synopsis

tcp.recvmsg

Values

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

Context

The process which receives a tcp message

Name

tcp.recvmsg.return — Receiving TCP message complete

Synopsis

```
tcp.recvmsg.return
```

Values

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

Context

The process which receives a tcp message

Name

tcp.disconnect — TCP socket disconnection

Synopsis

tcp.disconnect

Values

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

Context

The process which disconnects tcp

Name

tcp.disconnect.return — TCP socket disconnection complete

Synopsis

```
tcp.disconnect.return
```

Values

ret Error code (0: no error)

name Name of this probe

Context

The process which disconnects tcp

Name

tcp.setsockopt — Call to setsockopt

Synopsis

tcp.setsockopt

Values

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

Context

The process which calls setsockopt

Name

tcp.setsockopt.return — Return from setsockopt

Synopsis

```
tcp.setsockopt.return
```

Values

ret Error code (0: no error)

name Name of this probe

Context

The process which calls setsockopt

Name

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

Synopsis

udp.sendmsg

Values

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

Context

The process which sent a UDP message

Name

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

Synopsis

`udp.sendmsg.return`

Values

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

Context

The process which sent a UDP message

Name

udp.recvmsg — Fires whenever a UDP message is received

Synopsis

udp.recvmsg

Values

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

Context

The process which received a UDP message

Name

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

Synopsis

`udp.recvmsg.return`

Values

name The name of this probe

size Number of bytes received by the process

Context

The process which received a UDP message

Name

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

Synopsis

```
udp.disconnect
```

Values

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

Context

The process which requests a UDP disconnection

Name

`udp.disconnect.return` — UDP has been disconnected successfully

Synopsis

```
udp.disconnect.return
```

Values

ret Error code (0: no error)

name The name of this probe

Context

The process which requested a UDP disconnection

Name

`ip_ntop` — returns a string representation from an integer IP number

Synopsis

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

Arguments

addr the ip represented as an integer

Chapter 8. Socket Tapset

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

Name

`socket.send` — Message sent on a socket.

Synopsis

`socket.send`

Values

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

Context

The message sender

Name

socket.receive — Message received on a socket.

Synopsis

```
socket.receive
```

Values

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

Context

The message receiver

Name

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

Synopsis

`socket.sendmsg`

Values

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

Context

The message sender

Description

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

Name

`socket.sendmsg.return` — Return from `<command>socket.sendmsg</command>`.

Synopsis

`socket.sendmsg.return`

Values

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

Context

The message sender.

Description

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

Name

socket.recvmsg — Message being received on socket

Synopsis

`socket.recvmsg`

Values

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

Context

The message receiver.

Description

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

Name

`socket.recvmsg.return` — Return from Message being received on socket

Synopsis

`socket.recvmsg.return`

Values

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

Context

The message receiver.

Description

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

Name

socket.aio_write — Message send via sock_aio_write

Synopsis

```
socket.aio_write
```

Values

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

Context

The message sender

Description

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

Name

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

Synopsis

```
socket.aio_write.return
```

Values

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

Context

The message receiver.

Description

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

Name

socket.aio_read — Receiving message via sock_aio_read

Synopsis

```
socket.aio_read
```

Values

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

Context

The message sender

Description

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

Name

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

Synopsis

`socket.aio_read.return`

Values

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

Context

The message receiver.

Description

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

Name

socket.writev — Message sent via `socket_writev`

Synopsis

```
socket.writev
```

Values

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

Context

The message sender

Description

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

Name

`socket.writev.return` — Conclusion of message sent via `socket.writev`

Synopsis

`socket.writev.return`

Values

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

Context

The message receiver.

Description

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

Name

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

Synopsis

```
socket.readv
```

Values

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

Context

The message sender

Description

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

Name

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

Synopsis

`socket.readv.return`

Values

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

Context

The message receiver.

Description

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

Name

socket.create — Creation of a socket

Synopsis

```
socket.create
```

Values

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

Context

The requester (see requester variable)

Description

Fires at the beginning of creating a socket.

Name

`socket.create.return` — Return from Creation of a socket

Synopsis

```
socket.create.return
```

Values

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

Context

The requester (user process or kernel)

Description

Fires at the conclusion of creating a socket.

Name

socket.close — Close a socket

Synopsis

```
socket.close
```

Values

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

Context

The requester (user process or kernel)

Description

Fires at the beginning of closing a socket.

Name

`socket.close.return` — Return from closing a socket

Synopsis

`socket.close.return`

Values

name Name of this probe

Context

The requester (user process or kernel)

Description

Fires at the conclusion of closing a socket.

Name

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

Synopsis

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

Arguments

proto -- undescribed --

Name

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

Synopsis

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

Arguments

proto -- undescrbed --

Name

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

Synopsis

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

Arguments

family -- undescribed --

Name

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

Synopsis

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

Arguments

family -- undescribed --

Description

protocol family number.

Name

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

Synopsis

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

Arguments

state -- undscribed --

Name

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

Synopsis

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

Arguments

```
state    -- undescrbed --
```

Chapter 9. Process Tapset

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

Name

`process.create` — Fires whenever a new process is successfully created

Synopsis

```
process.create
```

Values

new_pid The PID of the newly created process

Context

Parent of the created process.

Description

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

Name

process.start — Starting new process

Synopsis

```
process.start
```

Values

None

Context

Newly created process.

Description

Fires immediately before a new process begins execution.

Name

process.exec — Attempt to exec to a new program

Synopsis

```
process.exec
```

Values

filename The path to the new executable

Context

The caller of exec.

Description

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

Name

`process.exec_complete` — Return from exec to a new program

Synopsis

```
process.exec_complete
```

Values

success A boolean indicating whether the exec was successful

errno The error number resulting from the exec

Context

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

Description

Fires at the completion of an exec call.

Name

`process.exit` — Exit from process

Synopsis

`process.exit`

Values

code The exit code of the process

Context

The process which is terminating.

Description

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

Name

process.release — Process released

Synopsis

```
process.release
```

Values

<i>pid</i>	PID of the process being released
<i>task</i>	A task handle to the process being released

Context

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

Description

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

Chapter 10. Signal Tapset

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

Name

signal.send — Signal being sent to a process

Synopsis

```
signal.send
```

Values

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

Context

The signal's sender.

Name

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

Synopsis

signal.send.return

Values

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

Context

The signal's sender. `<remark>(correct?)</remark>`

Description

Possible `<command>__group_send_sig_info</command>` and `<command>specific_send_sig_info</command>` return values are as follows;

`<command>0</command>` -- 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 `<command>sigqueue</command>` of the receiving process.

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

Possible `<command>send_group_sigqueue</command>` and `<command>send_sigqueue</command>` return values are as follows;

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

`<command>1</command>` -- The signal was ignored by the receiving process.

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

Name

signal.checkperm — Check being performed on a sent signal

Synopsis

signal.checkperm

Values

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

Name

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; default value is <command>signal.checkperm</command>

Name

signal.wakeup — Sleeping process being wakened for signal

Synopsis

signal.wakeup

Values

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

Name

`signal.check_ignored` — Checking to see signal is ignored

Synopsis

```
signal.check_ignored
```

Values

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

Name

`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; default value is `<command>signal.checkperm</command>`

Name

`signal.force_segv` — Forcing send of `<command>SIGSEGV</command>`

Synopsis

`signal.force_segv`

Values

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

Name

`signal.force_segv.return` — Forcing send of <command>SIGSEGV</command> complete

Synopsis

`signal.force_segv.return`

Values

retstr Return value as a string

name Name of the probe point; default value is <command>force_sigsegv</command>

Name

signal.syskill — Sending kill signal to a process

Synopsis

```
signal.syskill
```

Values

sig The specific signal sent to the process

pid The PID of the process receiving the signal

Name

signal.syskill.return — Sending kill signal completed

Synopsis

```
signal.syskill.return
```

Values

None

Name

signal.sys_tkill — Sending a kill signal to a thread

Synopsis

```
signal.sys_tkill
```

Values

<i>sig_name</i>	The specific signal sent to the process
<i>sig</i>	The specific signal sent to the process
<i>pid</i>	The PID of the process receiving the kill signal

Description

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

Name

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

Synopsis

```
signal.systkill.return
```

Values

None

Name

signal.sys_tgkill — Sending kill signal to a thread group

Synopsis

```
signal.sys_tgkill
```

Values

<i>sig_name</i>	A string representation of the signal
<i>sig</i>	The specific kill signal sent to the process
<i>pid</i>	The PID of the thread receiving the kill signal
<i>tgid</i>	The thread group ID of the thread receiving the kill signal

Description

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

Name

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

Synopsis

```
signal.sys_tgkill.return
```

Values

None

Name

`signal.send_sig_queue` — Queuing a signal to a process

Synopsis

```
signal.send_sig_queue
```

Values

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

Name

`signal.send_sig_queue.return` — Queuing a signal to a process completed

Synopsis

```
signal.send_sig_queue.return
```

Values

<i>retstr</i>	Return value as a string
---------------	--------------------------

Name

signal.pending — Examining pending signal

Synopsis

```
signal.pending
```

Values

<i>sigset_size</i>	The size of the user-space signal set
<i>sigset_add</i>	The address of the user-space signal set (<command>sigset_t</command>)

Description

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

Name

`signal.pending.return` — Examination of pending signal completed

Synopsis

`signal.pending.return`

Values

retstr Return value as a string

Name

`signal.handle` — Signal handler being invoked

Synopsis

`signal.handle`

Values

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

Name

`signal.handle.return` — Signal handler invocation completed

Synopsis

`signal.handle.return`

Values

retstr Return value as a string

Name

`signal.do_action` — Examining or changing a signal action

Synopsis

```
signal.do_action
```

Values

<i>sa_mask</i>	The new mask of the signal
<i>oldsigact_addr</i>	The address of the old <code><command>sigaction</command></code> struct associated with the signal
<i>sig</i>	The signal to be examined/changed
<i>sa_handler</i>	The new handler of the signal
<i>sigact_addr</i>	The address of the new <code><command>sigaction</command></code> struct associated with the signal

Name

`signal.do_action.return` — Examining or changing a signal action completed

Synopsis

```
signal.do_action.return
```

Values

<i>retstr</i>	Return value as a string
---------------	--------------------------

Name

signal.procmask — Examining or changing blocked signals

Synopsis

signal.procmask

Values

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

Name

`signal.flush` — Flusing all pending signals for a task

Synopsis

`signal.flush`

Values

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