SystemTap Tapset Reference Manual

William Cohen <wcohen@redhat.com>

SystemTap Tapset Reference Manual

by William Cohen

Table of Contents

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 the where the event occurred. These functions can provide information such as a backtrace where the event occured and the current register values for the processor.

Name

print_regs — Print a register dump.

Synopsis

```
print_regs()
```

Arguments

None

Name

execname — Execname of current processes

Synopsis

```
execname:string()
```

Arguments

None

Description

Return the name of the current process.

Name

pid — Process ID of current process

Synopsis

```
pid:long()
```

Arguments

None

Description

Return the id of the current process.

Name

tid — Thread ID of current process

Synopsis

```
tid:long()
```

Arguments

None

Description

Return the id of the current thread.

Name

ppid — Parent Process ID of current process

Synopsis

```
ppid:long()
```

Arguments

None

Description

Return the id of the parent process.

Name

pexecname — Execname of the parent process.

Synopsis

```
pexecname:string()
```

Arguments

None

Description

Return the name of the parent process.

Name

gid — Group ID of current process

Synopsis

```
gid:long()
```

Arguments

None

Description

Return the gid of the current process.

Name

egid — Effective gid of the current process.

Synopsis

```
egid:long()
```

Arguments

None

Description

Return the effective gid of the current process.

Name

uid — User ID of the current process.

Synopsis

```
uid:long()
```

Arguments

None

Description

Return the uid of the current process.

Name

euid — Effective User ID of the current process.

Synopsis

```
euid:long()
```

Arguments

None

Description

Return the effective uid of the current process.

Name

cpu — The current cpu number.

Synopsis

```
cpu:long()
```

Arguments

None

Description

Return the current cpu number.

Name

pp — Current probe point

Synopsis

```
pp:string()
```

Arguments

None

Description

Return the probe point associated with the currently running probe handler, including alias and wildcard expansion effects.

Name

registers_valid — Register information valid

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 — User Mode

Synopsis

```
user_mode:long()
```

Arguments

None

Description

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

Name

is_return — Is return probe

Synopsis

```
is_return:long()
```

Arguments

None

Description

Return 1 if the probe point is a return probe. <emphasis>Deprecated.</emphasis>

Name

target — Target pid

Synopsis

```
target:long()
```

Arguments

None

Description

Return the pid of the target process.

Name

stack_size — Size of kernel stack

Synopsis

```
stack_size:long()
```

Arguments

None

Description

Return the size of the kernel stack.

Name

stack_used — Current amount of kernel stack used

Synopsis

```
stack_used:long()
```

Arguments

None

Description

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

Name

stack_unused — Amount of kernel stack currently available

Synopsis

```
stack_unused:long()
```

Arguments

None

Description

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

Name

print_stack — Print out stack from string

Synopsis

```
print_stack(tk:string)
```

Arguments

tk -- undescribed --

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`. 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 — Function probed

Synopsis

```
probefunc:string()
```

Arguments

None

Description

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

Name

probemod — Module probed

Synopsis

```
probemod:string()
```

Arguments

None

Description

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

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. It may be truncated due to 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. <emphasis> Works only for return probes at this time.</emphasis>

Chapter 3. Timestamp Functions

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

Name

gettimeofday_ns — Number of nanoseconds since UNIX epoch.

Synopsis

```
gettimeofday_ns:long()
```

Arguments

None

Description

Return the number of nanoseconds since the UNIX epoch.

Name

gettimeofday_us — Number of microseconds since UNIX epoch.

Synopsis

```
gettimeofday_us:long()
```

Arguments

None

Description

Return the number of microseconds since the UNIX epoch.

Name

gettimeofday_ms — Number of milliseconds since UNIX epoch.

Synopsis

```
gettimeofday_ms:long()
```

Arguments

None

Description

Return the number of milliseconds since the UNIX epoch.

Name

gettimeofday_s — Number of seconds since UNIX epoch.

Synopsis

```
gettimeofday_s:long()
```

Arguments

None

Description

Return the number of seconds since the UNIX epoch.

Chapter 4. Memory Tapset

Name

vm_fault_contains — Test return value for page fault reason

Synopsis

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

Arguments

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

Name

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

Synopsis

```
probe 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

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

Synopsis

```
probe 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

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

Synopsis

```
probe 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

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

Synopsis

```
probe 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

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

Synopsis

```
probe vm.mmap
```

Values

length	The length of the memory segment
address	The requested address

Context

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

Name

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

Synopsis

```
probe vm.munmap
```

Values

length	The length of the memory segment
address	The requested address

Context

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

Name

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

Synopsis

```
probe vm.brk
```

Values

length	The length of the memory segment
address	The requested address

Context

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

Name

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

Synopsis

```
probe 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 the IO scheduler activities. It contains the following probe points:

Name

probe ioscheduler.elv_next_request — Retrieve request from request queue

Synopsis

```
probe ioscheduler.elv_next_request
```

Values

elevator_name	The elevator name

Name

probe ioscheduler.elv_next_request.return — Return from retrieving a request

Synopsis

```
probe ioscheduler.elv_next_request.return
```

Values

req_flags	Request flags
req	Address of the request
disk_major	Disk major number of the request
disk_minor	Disk minor number of the request

Name

probe ioscheduler.elv_add_request — Add a request into request queue

Synopsis

```
probe ioscheduler.elv_add_request
```

Values

req_flags	Request flags
req	Address of the request
disk_major	Disk major number of the request
elevator_name	The elevator name
disk_minor	Disk minor number of the request

Name

probe ioscheduler.elv_completed_request — Request is completed

Synopsis

```
probe ioscheduler.elv_completed_request
```

Values

req_flags	Request flags
req	Address of the request
disk_major	Disk major number of the request
elevator_name	The elevator name
disk_minor	Disk minor number of the request

Chapter 6. SCSI Tapset

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

Name

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

Synopsis

```
probe scsi.ioentry
```

Values

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

Name

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

Synopsis

```
probe scsi.iodispatching
```

Values

lun	The lun number
req_bufflen	The request buffer length
host_no	The host number
device_state	The current state of the device.
dev_id	The scsi device id
channel	The channel 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)
request_buffer	The request buffer address

Name

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

Synopsis

```
probe scsi.iodone
```

Values

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

Name

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

Synopsis

```
probe scsi.iocompleted
```

Values

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

Chapter 7. Networking Tapset

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

Name

probe netdev.receive — Data recieved from network device.

Synopsis

```
probe netdev.receive
```

Values

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

Name

probe netdev.transmit — Network device transmitting buffer

Synopsis

```
probe netdev.transmit
```

Values

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

Chapter 8. Socket Tapset

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

Name

probe socket.send — Message sent on a socket.

Synopsis

```
probe socket.send
```

Values

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

Context

The message sender

Name

probe socket.receive — Message received on a socket.

Synopsis

```
probe socket.receive
```

Values

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

Context

The message receiver

Name

probe socket.sendmsg.return — Return from Message being sent on socket

Synopsis

```
probe socket.sendmsg.return
```

Values

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

Context

The message sender.

Description

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

Name

probe socket.recvmsg — Message being received on socket

Synopsis

```
probe socket.recvmsg
```

Values

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

Context

The message receiver.

Description

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

Name

probe socket.aio_write — Message send via `sock_aio_write`

Synopsis

```
probe socket.aio_write
```

Values

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

Context

The message sender

Description

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

Name

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

Synopsis

```
probe socket.aio_write.return
```

Values

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

Name

probe socket.aio_read — Receiving message via `sock_aio_read`

Synopsis

```
probe socket.aio_read
```

Values

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

Context

The message sender

Description

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

Name

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

Synopsis

```
probe socket.aio_read.return
```

Values

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

Context

The message receiver.

Description

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

Name

probe socket.writev — Message sent via `socket_writev`

Synopsis

```
probe socket.writev
```

Values

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

Context

The message sender

Description

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

Name

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

Synopsis

```
probe socket.writev.return
```

Values

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

Name

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

Synopsis

```
probe socket.readv
```

Values

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

Context

The message sender

Description

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

Name

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

Synopsis

```
probe socket.readv.return
```

Values

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

Context

The message receiver.

Description

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

Name

probe socket.create — Creation of a socket

Synopsis

```
probe socket.create
```

Values

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

Context

The requester (see requester variable)

Description

Fires at the beginning of creating a socket.

Name

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

Synopsis

```
probe socket.create.return
```

Values

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

Context

The requester (user process or kernel)

Description

Fires at the conclusion of creating a socket.

Name

probe socket.close — Close a socket

Synopsis

```
probe socket.close
```

Values

protocol	Protocol value
flags	Socket flags value
name	Name of this probe
state	Socket state value
type	Socket type value
family	Protocol family value

Context

The requester (user process or kernel)

Description

Fires at the beginning of closing a socket.

Name

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

Synopsis

```
probe 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.

Chapter 9. TCP Tapset

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

Name

probe tcp.sendmsg — Sending a tcp message

Synopsis

```
probe tcp.sendmsg
```

Values

name	Name of this probe
size	Number of bytes to send
sock	Network socket

Context

The process which sends a tcp message

Name

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

Synopsis

```
probe tcp.sendmsg.return
```

Values

name Name of this probe

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

Context

The process which sends a tcp message

Name

probe tcp.recvmsg — Receiving TCP message

Synopsis

```
probe tcp.recvmsg
```

Values

name	Name of this probe
size	Number of bytes to be received
sock	Network socket

Context

The process which receives a tcp message

Name

probe tcp.recvmsg.return — Receiving TCP message complete

Synopsis

```
probe tcp.recvmsg.return
```

Values

name Name of this probe

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

Context

The process which receives a tcp message

Name

probe tcp.disconnect — TCP socket disconnection

Synopsis

```
probe tcp.disconnect
```

Values

flags	TCP flags (e.g. FIN, etc)
name	Name of this probe
sock	Network socket

Context

The process which disconnects tcp

Name

probe tcp.disconnect.return — TCP socket disconnection complete

Synopsis

```
probe tcp.disconnect.return
```

Values

ret Error code (0: no error)

name Name of this probe

Context

The process which disconnects tcp

Name

probe tcp.setsockopt — Call to `setsockopt`

Synopsis

```
probe tcp.setsockopt
```

Values

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

Context

The process which calls setsockopt

Name

probe tcp.setsockopt.return — Return from `setsockopt`

Synopsis

```
probe tcp.setsockopt.return
```

Values

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

Context

The process which calls setsockopt

Chapter 10. UDP Tapset

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

Name

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

Synopsis

```
probe udp.sendmsg
```

Values

name Name of this probe

size Number of bytes to send

sock Network socket

Context

The process which sends a udp message

Name

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

Synopsis

```
probe udp.sendmsg.return
```

Values

name Name of this probe

size Number of bytes sent

Context

The process which sends a udp message

Name

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

Synopsis

```
probe udp.recvmsg
```

Values

name Name of this probe

size Number of bytes received

sock Network socket

Context

The process which receives a udp message

Name

probe udp.recvmsg.return — An attempt to receive a UDP message received has been completed

Synopsis

```
probe udp.recvmsg.return
```

Values

name Name of this probe

size Number of bytes received

Context

The process which receives a udp message

Name

probe udp.disconnect — A process requests for UPD to be UDP disconnected

Synopsis

```
probe udp.disconnect
```

Values

flags	Flags (e.g. FIN, etc)
name	Name of this probe
sock	Network socket

Context

The process which disconnects UDP

Name

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

Synopsis

```
probe udp.disconnect.return
```

Values

ret Error code (0: no error)

name Name of this probe

Context

The process which disconnects udp

Chapter 11. Process Tapset

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

Name

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

Synopsis

```
probe 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

probe process.start — Starting new process

Synopsis

```
probe process.start
```

Values

None

Context

Newly created process.

Description

Fires immediately before a new process begins execution.

Name

probe process.exec — Attempt to exec to a new program

Synopsis

```
probe 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

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

Synopsis

```
probe 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

probe process.exit — Exit from process

Synopsis

```
probe 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

probe process.release — Process released

Synopsis

```
probe process.release
```

Values

pid PID of the process being released

task A task handle to the process being released

Context

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

Description

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

Chapter 12. Signal Tapset

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

Name

probe signal.send — Fires when a system call or kernel function sends a signal to a process.

Synopsis

```
probe signal.send
```

Values

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

Context

The signal's sender.

Name

probe signal.send.return — Fires when a signal sent to a process returns.

Synopsis

```
probe signal.send.return
```

Values

retstr The return value to either <command>__group_send_sig_info</command>, <command>specific_send_sig_info</command>, or <command>send_sigqueue</command>. Refer to the Description of this probe for more information about the return values of each function call.

send2queue Indicates whether the sent signal was sent to an existing <command>sigqueue</command>

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. <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 sucessfully 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 sucessfully 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

probe signal.checkperm — Fires when a permission check is performed on a sent signal

Synopsis

```
probe signal.checkperm
```

Values

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

Name

probe signal.wakeup — Wakes up a sleeping process, making it ready for new active signals

Synopsis

```
probe signal.wakeup
```

Values

resume Indicates whether to wake up a task in a <command>STOPPED</command> or <command>TRACED</command> state

state_mask A string representation indicating the mask of task states you wish to wake. Possible values are <command>TASK_INTERRUPTIBLE</command>, <command>TASK_STOPPED</command>, <command>TASK_TRACED</command>, and <command>TASK_INTERRUPTIBLE</command>.

pid_name Name of the process you wish to wake

sig_pid The PID of the process you wish to wake

Name

probe signal.check_ignored — Fires when a system call or kernel function checks whether a

Synopsis

```
probe signal.check_ignored
```

Values

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

Description

signal was ignored or not

Name

probe signal.force_segv — Fires when a system call, kernel function, or process sent a

Synopsis

```
probe signal.force_segv
```

Values

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

Description

<command>SIGSEGV</command> as a result of problems it encountered while handling a received signal

Name

probe signal.syskill — Fires when the kernel function <command>sys_kill</command>

Synopsis

```
probe signal.syskill
```

Values

sig The specific signal sent to the process

pid The PID of the process receiving the kill signal

Description

sends a kill signal to a process

Name

probe signal.sys_tkill — Fires when <command>tkill</command> sends a kill signal

Synopsis

```
probe signal.sys_tkill
```

Values

sig The specific signal sent to the process 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).

pid The PID of the process receiving the kill signal

Description

to a process that is part of a thread group

Name

probe signal.sys_tgkill — Fires when the kernel function <command>tgkill</command>

Synopsis

```
probe signal.sys_tgkill
```

Values

sig The specific kill signal sent to the process 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.

pid The PID of the thread receiving the kill signal

tgid The thread group ID of the thread receiving the kill signal

Description

sends a kill signal to a specific thread group

Name

probe signal.send_sig_queue — Fires when a signal is queued to a process

Synopsis

```
probe signal.send_sig_queue
```

Values

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

Name

probe signal.pending — Fires when the <command>SIGPENDING</command> system call is used;

Synopsis

```
probe signal.pending
```

Values

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

Description

this normally occurs when the <command>do_sigpending</command> kernel function is executed

Synopsis

<programlisting>long do_sigpending(void __user *set, unsigned long sigsetsize)</programlisting>

This probe is used to examine a set of signals pending for delivery to a specific thread.

Name

probe signal.handle — Fires when the signal handler is invoked

Synopsis

```
probe signal.handle
```

Values

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

Synopsis

<programlisting>static int handle_signal(unsigned long sig, siginfo_t *info, struct k_sigaction *ka, sigset_t *oldset, struct pt_regs * regs)</programlisting>

Name

probe signal.do_action — Initiates a trace when a thread is about to examine

Synopsis

```
probe signal.do_action
```

Values

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

Description

and change a signal action

Name

probe signal.procmask — Initiates a trace when a thread is about to examine and change blocked signals

Synopsis

```
probe signal.procmask
```

Values

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

Synopsis

<programlisting>int sigprocmask(int how, sigset_t *set, sigset_t *oldset)</programlisting>

Name

probe signal.flush — Fires when all pending signals for a task are flushed

Synopsis

```
probe signal.flush
```

Values

task	The task handler of the process performing the flush
pid_name	The name of the process associated with the task performing the flush
sig_pid	The PID of the process associated with the task performing the flush

Synopsis

<programlisting>void flush_signals(struct task_struct *t)</programlisting>