SystemTap Tapset Reference Manual

SystemTap

SystemTap Tapset Reference Manual

by SystemTap

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 where an event occurred. These functions can provide information such as a backtrace to 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 — 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

euid — Return the effective uid of a target process.

Synopsis

```
          euid: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. <emphasis>Deprecated.</emphasis>

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

uaddr — User space address of current running task. EXPERIMENTAL.

Synopsis

```
uaddr:long()
```

Arguments

None

Description

Returns the address in userspace that the current task was at when the probe occured. 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 `symdata`. Often the task will be in the VDSO where it entered the kernel. FIXME - need VDSO tracking support #10080.

Name

print_stack — Print out stack from string.

Synopsis

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

Arguments

stk
String with list of hexidecimal 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 <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

modname — Return the kernel module name loaded at the address.

Synopsis

```
modname:string(addr:long)
```

Arguments

addr
The address.

Description

Returns the module name associated with the given address if known. If not known it will return the string “<unknown>”. If the address was not in a kernel module, but in the kernel itself, then the string “kernel” will be returned.

Name

symname — Return the symbol associated with the given address.

Synopsis

```
symname:string(addr:long)
```

Arguments

addr
The address to translate.

Description

Returns the (function) symbol name associated with the given address if known. If not known it will return the hex string representation of addr.

Name

symdata — Return the 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, plus the module name (between brackets) and the offset inside the module, plus the size of the symbol function. If any element is not known it will be ommitted and if the symbol name is unknown it will return the hex string for the given address.

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. <emphasis> Works only for return probes at this time.</emphasis>

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

value
The fault_type returned by vm.page_fault.return

test
The type of fault to test for (VM_FAULT_OOM 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

elevator_name
: 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

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

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

Synopsis

```
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 type of I/O elevator currently enabled

disk_minor
: Disk minor number of the request

Name

ioscheduler.elv_completed_request — Fires when a request is completed

Synopsis

```
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 type of I/O elevator currently enabled

disk_minor
: 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

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

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

Synopsis

```
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

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

Synopsis

```
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

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

Synopsis

```
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 the network device and protocol layers.

Name

netdev.receive — Data recieved from network device.

Synopsis

```
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

netdev.transmit — Network device transmitting buffer

Synopsis

```
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.

Name

tcp.sendmsg — Sending a tcp message

Synopsis

```
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

tcp.sendmsg.return — Sending TCP message is done

Synopsis

```
tcp.sendmsg.return
```

Values

name
Name of this probe

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

Context

The process which sends a tcp message

Name

tcp.recvmsg — Receiving TCP message

Synopsis

```
tcp.recvmsg
```

Values

saddr
A string representing the source IP address

daddr
A string representing the destination IP address

name
Name of this probe

sport
TCP source port

dport
TCP destination port

size
Number of bytes to be received

sock
Network socket

Context

The process which receives a tcp message

Name

tcp.recvmsg.return — Receiving TCP message complete

Synopsis

```
tcp.recvmsg.return
```

Values

saddr
A string representing the source IP address

daddr
A string representing the destination IP address

name
Name of this probe

sport
TCP source port

dport
TCP destination port

size
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

saddr
A string representing the source IP address

daddr
A string representing the destination IP address

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

name
Name of this probe

sport
TCP source port

dport
TCP destination port

sock
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

`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

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

tcp.receive — Called when a TCP packet is received

Synopsis

```
tcp.receive
```

Values

urg
TCP URG flag

psh
TCP PSH flag

rst
TCP RST flag

dport
TCP destination port

saddr
A string representing the source IP address

daddr
A string representing the destination IP address

ack
TCP ACK flag

syn
TCP SYN flag

fin
TCP FIN flag

sport
TCP source port

Name

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

Synopsis

```
udp.sendmsg
```

Values

name
The name of this probe

size
Number of bytes sent by the process

sock
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

name
The name of this probe

size
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

name
The name of this probe

size
Number of bytes received by the process

sock
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

flags
Flags (e.g. FIN, etc)

name
The name of this probe

sock
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

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

socket.receive — Message received on a socket.

Synopsis

```
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

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

Synopsis

```
socket.sendmsg
```

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 the `sock_sendmsg` function

Name

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

Synopsis

```
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

socket.recvmsg — Message being received on socket

Synopsis

```
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

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

Synopsis

```
socket.recvmsg.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_recvmsg` function.

Name

socket.aio_write — Message send via `sock_aio_write`

Synopsis

```
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

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

Synopsis

```
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

socket.aio_read — Receiving message via `sock_aio_read`

Synopsis

```
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

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

Synopsis

```
        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

socket.writev — Message sent via `socket_writev`

Synopsis

```
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

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

Synopsis

```
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

socket.readv — Receiving a message via `sock_readv`

Synopsis

```
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

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

Synopsis

```
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

socket.create — Creation of a socket

Synopsis

```
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

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

Synopsis

```
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

socket.close — Close a socket

Synopsis

```
        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

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
The protocol number.

Name

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

Synopsis

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

Arguments

proto
The protocol name.

Name

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

Synopsis

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

Arguments

family
The family number.

Name

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

Synopsis

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

Arguments

family
The family name.

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
The state number.

Name

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.

Chapter 9. Kernel Process Tapset

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

Name

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

Synopsis

```
kprocess.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

kprocess.start — Starting new process

Synopsis

```
        kprocess.start
```

Values

None

Context

Newly created process.

Description

Fires immediately before a new process begins execution.

Name

kprocess.exec — Attempt to exec to a new program

Synopsis

```
kprocess.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

kprocess.exec_complete — Return from exec to a new program

Synopsis

```
kprocess.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

kprocess.exit — Exit from process

Synopsis

```
        kprocess.exit
```

Values

code
The exit code of the process

Context

The process which is terminating.

Description

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

Name

kprocess.release — Process released

Synopsis

```
kprocess.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 kprocess.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

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

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

sig_pid
The PID of the process receiving the signal

pid_name
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

retstr
The return value to either <command>__group_send_sig_info</command>, <command>specific_send_sig_info</command>, or <command>send_sigqueue</command>

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

signal.checkperm — Check being performed on a sent signal

Synopsis

```
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

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

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 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 to wake

sig_pid
: The PID of the process to wake

Name

signal.check_ignored — Checking to see signal is ignored

Synopsis

```
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

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

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

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

sig_name
The specific signal sent to the process

sig
The specific signal sent to the process

pid
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

sig_name
A string representation of the signal

sig
The specific kill signal sent to the process

pid
The PID of the thread receiving the kill signal

tgid
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

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

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

Synopsis

```
signal.send_sig_queue.return
```

Values

retstr
Return value as a string

Name

signal.pending — Examining pending signal

Synopsis

```
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 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

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

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

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

Name

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

Synopsis

```
signal.do_action.return
```

Values

retstr
Return value as a string

Name

signal.procmask — Examining or changing blocked signals

Synopsis

```
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

Name

signal.flush — Flusing all pending signals for a task

Synopsis

```
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