Red Hat Enterprise Linux 7 SystemTap Tapset Reference

For SystemTap in Red Hat Enterprise Linux 7

Vladimír Slávik
Red Hat Customer Content Services
vslavik@redhat.com

Robert Krátký
Red Hat Customer Content Services

William Cohen
Red Hat Software Engineering

Don Domingo
Red Hat Customer Content Services

Jacquelynn East
Red Hat Customer Content Services

Red Hat Enterprise Linux Documentation
Abstract

The Tapset Reference Guide describes the most common tapset definitions users can apply to SystemTap scripts.
Table of Contents

CHAPTER 1. INTRODUCTION ................................................................. 52
  1.1. DOCUMENTATION GOALS ...................................................... 52

CHAPTER 2. TAPSET DEVELOPMENT GUIDELINES ............................... 53.
  2.1. WRITING GOOD TAPSETS ................................................... 53
  2.2. ELEMENTS OF A TAPSET ................................................... 54

CHAPTER 3. CONTEXT FUNCTIONS .................................................... 57.
  NAME ................................................................. 57
  SYNOPSIS .......................................................... 57
  ARGUMENTS ...................................................... 57
  DESCRIPTION ................................................... 57
  NAME ................................................................. 57
  SYNOPSIS .......................................................... 57
  ARGUMENTS ...................................................... 57
  DESCRIPTION ................................................... 57
  NAME ................................................................. 57
  SYNOPSIS .......................................................... 57
  ARGUMENTS ...................................................... 57
  DESCRIPTION ................................................... 57
  NAME ................................................................. 57
  SYNOPSIS .......................................................... 57
  ARGUMENTS ...................................................... 57
  DESCRIPTION ................................................... 57
  NAME ................................................................. 57
  SYNOPSIS .......................................................... 57
  ARGUMENTS ...................................................... 57
  DESCRIPTION ................................................... 57
  NAME ................................................................. 57
  SYNOPSIS .......................................................... 57
  ARGUMENTS ...................................................... 57
  DESCRIPTION ................................................... 57
  NAME ................................................................. 57
  SYNOPSIS .......................................................... 57
  ARGUMENTS ...................................................... 57
  DESCRIPTION ................................................... 57
  NAME ................................................................. 57
  SYNOPSIS .......................................................... 57
  ARGUMENTS ...................................................... 57
  DESCRIPTION ................................................... 57
  NAME ................................................................. 57
  SYNOPSIS .......................................................... 60
  ARGUMENTS ...................................................... 60
  DESCRIPTION ................................................... 60
  NAME ................................................................. 60
  SYNOPSIS .......................................................... 60
  ARGUMENTS ...................................................... 60
  DESCRIPTION ................................................... 60
  NAME ................................................................. 60
  SYNOPSIS .......................................................... 60
  ARGUMENTS ...................................................... 60
  DESCRIPTION ................................................... 60
### Table of Contents

<table>
<thead>
<tr>
<th>SYNOPSIS</th>
<th>ARGUMENTS</th>
<th>DESCRIPTION</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>65</td>
<td></td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>65</td>
<td></td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>65</td>
<td></td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>65</td>
<td></td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>65</td>
<td></td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>65</td>
<td></td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>65</td>
<td></td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td></td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td></td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td></td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td></td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td></td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td></td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td></td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td></td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td></td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td></td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td></td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>67</td>
<td></td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>67</td>
<td></td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>67</td>
<td></td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>67</td>
<td></td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>67</td>
<td></td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>67</td>
<td></td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>67</td>
<td></td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>67</td>
<td></td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>67</td>
<td></td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>67</td>
<td></td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>67</td>
<td></td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>68</td>
<td></td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>68</td>
<td></td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>68</td>
<td></td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>68</td>
<td></td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>68</td>
<td></td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>68</td>
<td></td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>68</td>
<td></td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>68</td>
<td></td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>68</td>
<td></td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>68</td>
<td></td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>68</td>
<td></td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>69</td>
<td></td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>69</td>
<td></td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>69</td>
<td></td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>69</td>
<td></td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>69</td>
<td></td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>69</td>
<td></td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>69</td>
<td></td>
<td>69</td>
</tr>
</tbody>
</table>
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
Table of Contents

NAME 93
SYNOPSIS 94
ARGUMENTS 94
DESCRIPTION 94
NAME 94
SYNOPSIS 94
ARGUMENTS 94
DESCRIPTION 94
NAME 94
SYNOPSIS 94
ARGUMENTS 94
DESCRIPTION 94
NAME 95
SYNOPSIS 95
ARGUMENTS 95
DESCRIPTION 95
NAME 95
SYNOPSIS 95
ARGUMENTS 95
DESCRIPTION 95
NAME 95
SYNOPSIS 95
ARGUMENTS 95
DESCRIPTION 95
NAME 95
SYNOPSIS 95
ARGUMENTS 95
DESCRIPTION 95
NAME 96
SYNOPSIS 96
ARGUMENTS 96
DESCRIPTION 96
NAME 96
SYNOPSIS 96
ARGUMENTS 96
DESCRIPTION 96
NAME 96
SYNOPSIS 96
ARGUMENTS 96
DESCRIPTION 96
NAME 96
SYNOPSIS 96
ARGUMENTS 96
DESCRIPTION 96
NAME 97
SYNOPSIS 97
ARGUMENTS 97
DESCRIPTION 97
NAME 97
SYNOPSIS 97
ARGUMENTS 97
DESCRIPTION 97
NAME 97
SYNOPSIS 97
ARGUMENTS 97
DESCRIPTION 97
NAME 97
SYNOPSIS 97
ARGUMENTS 97
DESCRIPTION 97
NAME 97
SYNOPSIS 97
ARGUMENTS 97
DESCRIPTION 97
NAME 98
NOTE 98
NAME 98
SYNOPSIS 98
ARGUMENTS 98
DESCRIPTION 98
NAME 98
SYNOPSIS 98
ARGUMENTS 98
DESCRIPTION 98
CHAPTER 4. TIMESTAMP FUNCTIONS
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Description</th>
<th>Name</th>
<th>Synopsis</th>
<th>Arguments</th>
<th>Description</th>
<th>Name</th>
<th>Synopsis</th>
<th>Arguments</th>
<th>Description</th>
<th>Name</th>
<th>Synopsis</th>
<th>Arguments</th>
<th>Description</th>
<th>Name</th>
<th>Synopsis</th>
<th>Arguments</th>
<th>Description</th>
<th>Name</th>
<th>Synopsis</th>
</tr>
</thead>
<tbody>
<tr>
<td>104</td>
<td></td>
<td>104</td>
<td></td>
<td>104</td>
<td></td>
<td>104</td>
<td></td>
<td>104</td>
<td></td>
<td>104</td>
<td></td>
<td>104</td>
<td></td>
<td>104</td>
<td></td>
<td>105</td>
<td></td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
<td>105</td>
<td></td>
<td>105</td>
<td></td>
<td>105</td>
<td></td>
<td>105</td>
<td></td>
<td>105</td>
<td></td>
<td>105</td>
<td></td>
<td>105</td>
<td></td>
<td>106</td>
<td></td>
<td>106</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td></td>
<td>106</td>
<td></td>
<td>106</td>
<td></td>
<td>106</td>
<td></td>
<td>106</td>
<td></td>
<td>106</td>
<td></td>
<td>106</td>
<td></td>
<td>106</td>
<td></td>
<td>106</td>
<td></td>
<td>106</td>
<td></td>
</tr>
<tr>
<td>106</td>
<td></td>
<td>106</td>
<td></td>
<td>106</td>
<td></td>
<td>106</td>
<td></td>
<td>106</td>
<td></td>
<td>106</td>
<td></td>
<td>106</td>
<td></td>
<td>106</td>
<td></td>
<td>107</td>
<td></td>
<td>107</td>
<td></td>
</tr>
<tr>
<td>107</td>
<td></td>
<td>107</td>
<td></td>
<td>107</td>
<td></td>
<td>107</td>
<td></td>
<td>107</td>
<td></td>
<td>107</td>
<td></td>
<td>107</td>
<td></td>
<td>107</td>
<td></td>
<td>107</td>
<td></td>
<td>107</td>
<td></td>
</tr>
<tr>
<td>107</td>
<td></td>
<td>107</td>
<td></td>
<td>107</td>
<td></td>
<td>107</td>
<td></td>
<td>107</td>
<td></td>
<td>107</td>
<td></td>
<td>107</td>
<td></td>
<td>107</td>
<td></td>
<td>107</td>
<td></td>
<td>107</td>
<td></td>
</tr>
<tr>
<td>107</td>
<td></td>
<td>107</td>
<td></td>
<td>107</td>
<td></td>
<td>107</td>
<td></td>
<td>107</td>
<td></td>
<td>107</td>
<td></td>
<td>107</td>
<td></td>
<td>107</td>
<td></td>
<td>108</td>
<td></td>
<td>108</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 5. TIME UTILITY FUNCTIONS

NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION

CHAPTER 6. SHELL COMMAND FUNCTIONS
<table>
<thead>
<tr>
<th>Chapter 8. Task Time Tapset</th>
<th>129</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>129</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>129</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>129</td>
</tr>
<tr>
<td>NAME</td>
<td>129</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>129</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>129</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>130</td>
</tr>
<tr>
<td>NAME</td>
<td>130</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>130</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>130</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>130</td>
</tr>
<tr>
<td>NAME</td>
<td>130</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>130</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>130</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>130</td>
</tr>
<tr>
<td>NAME</td>
<td>130</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>130</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>130</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>130</td>
</tr>
<tr>
<td>NAME</td>
<td>130</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>130</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>130</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>130</td>
</tr>
<tr>
<td>NAME</td>
<td>130</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>130</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>130</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>130</td>
</tr>
<tr>
<td>NAME</td>
<td>130</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>130</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>130</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>130</td>
</tr>
<tr>
<td>NAME</td>
<td>130</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>130</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>130</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>130</td>
</tr>
<tr>
<td>NAME</td>
<td>131</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>131</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>131</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>131</td>
</tr>
<tr>
<td>NAME</td>
<td>131</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>131</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>131</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>131</td>
</tr>
<tr>
<td>NAME</td>
<td>131</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>131</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>131</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>131</td>
</tr>
<tr>
<td>NAME</td>
<td>131</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>131</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>131</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>131</td>
</tr>
<tr>
<td>NAME</td>
<td>131</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>131</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>131</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>131</td>
</tr>
<tr>
<td>NAME</td>
<td>131</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>131</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>131</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>131</td>
</tr>
<tr>
<td>NAME</td>
<td>131</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>131</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>131</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>131</td>
</tr>
<tr>
<td>NAME</td>
<td>131</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>131</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>131</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>131</td>
</tr>
<tr>
<td>NAME</td>
<td>131</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>131</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>131</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>131</td>
</tr>
<tr>
<td>NAME</td>
<td>131</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>131</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>131</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>131</td>
</tr>
<tr>
<td>NAME</td>
<td>131</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>131</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>131</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>131</td>
</tr>
<tr>
<td>NAME</td>
<td>131</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>131</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>131</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>131</td>
</tr>
<tr>
<td>NAME</td>
<td>131</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>131</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>131</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>131</td>
</tr>
<tr>
<td>NAME</td>
<td>131</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>131</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>131</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>131</td>
</tr>
<tr>
<td>NAME</td>
<td>131</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>131</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>131</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>131</td>
</tr>
<tr>
<td>NAME</td>
<td>131</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>131</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>131</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>131</td>
</tr>
<tr>
<td>NAME</td>
<td>131</td>
</tr>
<tr>
<td>SYNOPSIS</td>
<td>131</td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>131</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>131</td>
</tr>
</tbody>
</table>
CHAPTER 10. IO SCHEDULER AND BLOCK IO TAPSET

NAME

SYNOPSIS

VALUES

CONTEXT

NAME

SYNOPSIS

VALUES

CONTEXT

NAME

SYNOPSIS

VALUES

CONTEXT

NAME

SYNOPSIS

VALUES

CONTEXT

NAME

SYNOPSIS

VALUES

CONTEXT

NAME

SYNOPSIS

VALUES

CONTEXT

NAME

SYNOPSIS

VALUES

CONTEXT

NAME

SYNOPSIS

VALUES

CONTEXT

NAME

SYNOPSIS

VALUES

CONTEXT

NAME

SYNOPSIS

VALUES

CONTEXT

NAME

SYNOPSIS

VALUES

CONTEXT

NAME

SYNOPSIS

VALUES

CONTEXT
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Name</th>
<th>Synopsis</th>
<th>Values</th>
<th>Description</th>
<th>Name</th>
<th>Synopsis</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 13. INTERRUPT REQUEST (IRQ) TAPSET ................................. 170.

NAME  170
SYNOPSIS  170
VALUES  170
NAME  171
SYNOPSIS  171
VALUES  171
NAME  172
SYNOPSIS  172
VALUES  172
NAME  172
SYNOPSIS  172
VALUES  172
NAME  173
SYNOPSIS  173
VALUES  173
NAME  173
SYNOPSIS  173
VALUES  173
CHAPTER 14. NETWORKING TAPSET

NAME
SYNOPSIS
VALUES
NAME
SYNOPSIS
VALUES
NAME
SYNOPSIS
VALUES
NAME
SYNOPSIS
VALUES
NAME
SYNOPSIS
VALUES
NAME
SYNOPSIS
VALUES
NAME
SYNOPSIS
VALUES
NAME
SYNOPSIS
VALUES
NAME
SYNOPSIS
VALUES
NAME
SYNOPSIS
VALUES
NAME
SYNOPSIS
VALUES
NAME
SYNOPSIS
VALUES
NAME
SYNOPSIS
VALUES
NAME
SYNOPSIS
VALUES
NAME
SYNOPSIS
VALUES
NAME
SYNOPSIS
VALUES
NAME
SYNOPSIS
VALUES
NAME
SYNOPSIS
VALUES
NAME
SYNOPSIS
VALUES
NAME
SYNOPSIS
VALUES
NAME
SYNOPSIS
VALUES
NAME
SYNOPSIS
VALUES
NAME
SYNOPSIS
VALUES
NAME
SYNOPSIS
VALUES
NAME
SYNOPSIS
VALUES
NAME
SYNOPSIS
VALUES
CHAPTER 15. SOCKET TAPSET
CHAPTER 18. SIGNAL TAPSET

NAME 276
SYNOPSIS 276
ARGUMENTS 276
NAME 276
SYNOPSIS 276
ARGUMENTS 276
NAME 276
SYNOPSIS 276
ARGUMENTS 277
NAME 277
SYNOPSIS 277
ARGUMENTS 277
DESCRIPTION 277
NAME 277
SYNOPSIS 277
ARGUMENTS 277
NAME 278
SYNOPSIS 278
<table>
<thead>
<tr>
<th>NAME</th>
<th>SYNOPSIS</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SYNOPSIS 286
VALUES 286
CONTEXT 287
DESCRIPTION 287
NAME 287
SYNOPSIS 287
VALUES 287
NAME 288
SYNOPSIS 288
VALUES 288
NAME 288
SYNOPSIS 288
VALUES 288
DESCRIPTION 289
NAME 289
SYNOPSIS 289
VALUES 289
NAME 289
SYNOPSIS 289
VALUES 289
DESCRIPTION 290
NAME 290
SYNOPSIS 290
VALUES 290
NAME 290
SYNOPSIS 291
VALUES 291
NAME 291
SYNOPSIS 291
VALUES 291
NAME 291
SYNOPSIS 291
VALUES 291
CHAPTER 19. ERRNO TAPSET

CHAPTER 20. RLIMIT TAPSET

CHAPTER 20. RLIMIT TAPSET 295.
ARGUMENTS 306
DESCRIPTION 306
NAME 306
SYNOPSIS 306
ARGUMENTS 306
DESCRIPTION 306
STATISTICS FOR THE GIVEN QUEUE 306
NAME 307
SYNOPSIS 307
ARGUMENTS 307
DESCRIPTION 307
NAME 307
SYNOPSIS 307
ARGUMENTS 307
DESCRIPTION 307
NAME 307
SYNOPSIS 307
ARGUMENTS 307
DESCRIPTION 307
NAME 307
SYNOPSIS 307
ARGUMENTS 307
DESCRIPTION 307
NAME 307
SYNOPSIS 307
ARGUMENTS 307
DESCRIPTION 307
NAME 307
SYNOPSIS 307
ARGUMENTS 308
DESCRIPTION 308
NAME 308
SYNOPSIS 308
ARGUMENTS 308
DESCRIPTION 308
NAME 308
SYNOPSIS 308
ARGUMENTS 308
DESCRIPTION 308
NAME 308
SYNOPSIS 308
ARGUMENTS 308
DESCRIPTION 308
NAME 308
SYNOPSIS 308
ARGUMENTS 308
DESCRIPTION 308
NAME 309
SYNOPSIS 309
ARGUMENTS 309
DESCRIPTION 309
NAME 309
SYNOPSIS 309
ARGUMENTS 309
DESCRIPTION 309
NAME 309
SYNOPSIS 309
ARGUMENTS 309
DESCRIPTION 309
NAME 310
SYNOPSIS 310
ARGUMENTS 310
CHAPTER 25. RANDOM FUNCTIONS TAPSET ............................... 310.
NAME 310
SYNOPSIS 310
ARGUMENTS 310
CHAPTER 26. STRING AND DATA RETRIEVING FUNCTIONS TAPSET .... 311.
NAME 311
SYNOPSIS 311
ARGUMENTS 311
DESCRIPTION 311
NAME 311
SYNOPSIS 311
ARGUMENTS 311
DESCRIPTION 311
NAME 311
SYNOPSIS 311
ARGUMENTS 311
DESCRIPTION 311
NAME 311
SYNOPSIS 311
ARGUMENTS 312
DESCRIPTION 312
NAME 312
SYNOPSIS 312
ARGUMENTS 312
CHAPTER 28. GURU TAPSETS .................................................................................. 337.

NAME 337
SYNOPSIS 337
ARGUMENTS 337
DESCRIPTION 337
NAME 337
SYNOPSIS 337
ARGUMENTS 337
DESCRIPTION 337
NAME 337
SYNOPSIS 337
ARGUMENTS 337
DESCRIPTION 337
NAME 337
SYNOPSIS 337
ARGUMENTS 337
DESCRIPTION 337
NAME 337
SYNOPSIS 337
ARGUMENTS 337
DESCRIPTION 337


NAME 339
SYNOPSIS 339
ARGUMENTS 339
DESCRIPTION 339
NAME 339
SYNOPSIS 339
ARGUMENTS 339
DESCRIPTION 339
NAME 339
SYNOPSIS 339
ARGUMENTS 339
DESCRIPTION 339
NAME 339
SYNOPSIS 339
ARGUMENTS 339
DESCRIPTION 339
NAME 339
SYNOPSIS 340
ARGUMENTS 340
DESCRIPTION 340
NAME 340
SYNOPSIS 340
CHAPTER 30. UTILITY FUNCTIONS FOR USING ANSI CONTROL CHARS IN LOGS

ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
CHAPTER 31. SYSTEMTAP TRANSLATOR TAPSET  .................................................. 350.

NAME  350
SYNOPSIS  350
VALUES  350
DESCRIPTION  350
NAME  350
SYNOPSIS  350
VALUES  350
DESCRIPTION  350
NAME  350
SYNOPSIS  350
VALUES  350
DESCRIPTION  350
NAME  351
SYNOPSIS  351
VALUES  351
DESCRIPTION  351
CHAPTER 32. NETWORK FILE STORAGE TAPSETS ................................................................. 361.

NAME .............................................. 361
SYNOPSIS ........................................ 361
ARGUMENTS ...................................... 361
DESCRIPTION .................................. 361
NAME .............................................. 361
SYNOPSIS ........................................ 361
VALUES .......................................... 361
DESCRIPTION .................................. 362
<table>
<thead>
<tr>
<th>NAME</th>
<th>SYNOPSIS</th>
<th>VALUES</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>362</td>
<td>362</td>
<td>362</td>
<td>363</td>
</tr>
<tr>
<td>363</td>
<td>363</td>
<td>363</td>
<td>363</td>
</tr>
<tr>
<td>363</td>
<td>364</td>
<td>364</td>
<td>364</td>
</tr>
<tr>
<td>364</td>
<td>365</td>
<td>365</td>
<td>365</td>
</tr>
<tr>
<td>365</td>
<td>366</td>
<td>366</td>
<td>366</td>
</tr>
<tr>
<td>366</td>
<td>367</td>
<td>367</td>
<td>367</td>
</tr>
<tr>
<td>367</td>
<td>368</td>
<td>368</td>
<td>368</td>
</tr>
<tr>
<td>368</td>
<td>369</td>
<td>369</td>
<td>369</td>
</tr>
<tr>
<td>369</td>
<td>370</td>
<td>370</td>
<td>370</td>
</tr>
<tr>
<td>370</td>
<td>371</td>
<td>371</td>
<td>371</td>
</tr>
<tr>
<td>371</td>
<td>372</td>
<td>372</td>
<td>372</td>
</tr>
<tr>
<td>372</td>
<td>373</td>
<td>373</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 33. SPECULATION

NAME
SYNOPSIS
ARGUMENTS
DESCRIPTION
NAME
SYNOPSIS
ARGUMENTS
NAME
SYNOPSIS
ARGUMENTS
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. This instrumentation uses probe points and functions provided in the tapset library.

Simply put, tapsets are scripts that encapsulate knowledge about a kernel subsystem into pre-written probes and functions that can be used by other scripts. Tapsets are analogous to libraries for C programs. They hide the underlying details of a kernel area while exposing the key information needed to manage and monitor that aspect of the kernel. They are typically developed by kernel subject-matter experts.

A tapset exposes the high-level data and state transitions of a subsystem. For the most part, good tapset developers assume that SystemTap users know little to nothing about the kernel subsystem's low-level details. As such, tapset developers write tapsets that help ordinary SystemTap users write meaningful and useful SystemTap scripts.

1.1. DOCUMENTATION GOALS

This guide aims to document SystemTap's most useful and common tapset entries; it also contains guidelines on proper tapset development and documentation. The tapset definitions contained in this guide are extracted automatically from properly-formatted comments in the code of each tapset file. As such, any revisions to the definitions in this guide should be applied directly to their respective tapset file.
CHAPTER 2. TAPSET DEVELOPMENT GUIDELINES

This chapter describes the upstream guidelines on proper tapset documentation. It also contains information on how to properly document your tapsets, to ensure that they are properly defined in this guide.

2.1. WRITING GOOD TAPSETS

The first step to writing good tapsets is to create a simple model of your subject area. For example, a model of the process subsystem might include the following:

**Key Data**
- process ID
- parent process ID
- process group ID

**State Transitions**
- forked
- exec’d
- running
- stopped
- terminated

*NOTE*
Both lists are examples, and are not meant to represent a complete list.

Use your subsystem expertise to find probe points (function entries and exits) that expose the elements of the model, then define probe aliases for those points. Be aware that some state transitions can occur in more than one place. In those cases, an alias can place a probe in multiple locations.

For example, process execs can occur in either the `do_execve()` or the `compat_do_execve()` functions. The following alias inserts probes at the beginning of those functions:

```plaintext
probe kprocess.exec = kernel.function("do_execve"),
kernel.function("compat_do_execve")
{probe body}
```

Try to place probes on stable interfaces (i.e., functions that are unlikely to change at the interface level) whenever possible. This will make the tapset less likely to break due to kernel changes. Where kernel version or architecture dependencies are unavoidable, use preprocessor conditionals (see the `stap(1)` man page for details).

Fill in the probe bodies with the key data available at the probe points. Function entry probes can access the entry parameters specified to the function, while exit probes can access the entry parameters and the return value. Convert the data into meaningful forms where appropriate (e.g., bytes to kilobytes, state
values to strings, etc).

You may need to use auxiliary functions to access or convert some of the data. Auxiliary functions often use embedded C to do things that cannot be done in the SystemTap language, like access structure fields in some contexts, follow linked lists, etc. You can use auxiliary functions defined in other tapsets or write your own.

In the following example, `copy_process()` returns a pointer to the `task_struct` for the new process. Note that the process ID of the new process is retrieved by calling `task_pid()` and passing it the `task_struct` pointer. In this case, the auxiliary function is an embedded C function defined in `task.stp`.

```plaintext
probe kprocess.create = kernel.function("copy_process").return
{
    task = $return
    new_pid = task_pid(task)
}
```

It is not advisable to write probes for every function. Most SystemTap users will not need or understand them. Keep your tapsets simple and high-level.

## 2.2. ELEMENTS OF A TAPSET

The following sections describe the most important aspects of writing a tapset. Most of the content herein is suitable for developers who wish to contribute to SystemTap's upstream library of tapsets.

### 2.2.1. Tapset Files

Tapset files are stored in `src/tapset/` of the SystemTap GIT directory. Most tapset files are kept at that level. If you have code that only works with a specific architecture or kernel version, you may choose to put your tapset in the appropriate subdirectory.

Installed tapsets are located in `/usr/share/systemtap/tapset/` or `/usr/local/share/systemtap/tapset/`.

Personal tapsets can be stored anywhere. However, to ensure that SystemTap can use them, use `-I tapset_directory` to specify their location when invoking `stap`.

### 2.2.2. Namespace

Probe alias names should take the form `tapset_name.probe_name`. For example, the probe for sending a signal could be named `signal.send`.

Global symbol names (probes, functions, and variables) should be unique across all tapsets. This helps avoid namespace collisions in scripts that use multiple tapsets. To ensure this, use tapset-specific prefixes in your global symbols.

Internal symbol names should be prefixed with an underscore (`_`).

### 2.2.3. Comments and Documentation

All probes and functions should include comment blocks that describe their purpose, the data they provide, and the context in which they run (e.g. interrupt, process, etc). Use comments in areas where your intent may not be clear from reading the code.
Note that specially-formatted comments are automatically extracted from most tapsets and included in this guide. This helps ensure that tapset contributors can write their tapset and document it in the same place. The specified format for documenting tapsets is as follows:

```c
/**
 * probe tapset.name - Short summary of what the tapset does.
 * @argument: Explanation of argument.
 * @argument2: Explanation of argument2. Probes can have multiple arguments.
 *
 * Context:
 * A brief explanation of the tapset context.
 * Note that the context should only be 1 paragraph short.
 *
 * Text that will appear under "Description."
 *
 * A new paragraph that will also appear under the heading "Description".
 *
 * Header:
 * A paragraph that will appear under the heading "Header".
**/
```

For example:

```c
/**
 * probe vm.write_shared_copy - Page copy for shared page write.
 * @address: The address of the shared write.
 * @zero: Boolean indicating whether it is a zero page
 * (can do a clear instead of a copy).
 *
 * Context:
 * The process attempting the write.
 *
 * Fires when a write to a shared page requires a page copy. This is always preceded by a `vm.shared_write`.
**/
```

To override the automatically-generated Synopsis content, use:

```c
* Synopsis:
  * New Synopsis string
```

For example:

```c
/**
 * probe signal.handle - Fires when the signal handler is invoked
 * @sig: The signal number that invoked the signal handler
 *
 * Synopsis:
 * <programlisting>static int handle_signal(unsigned long sig, siginfo_t *info, struct k_sigaction *ka,
 * sigset_t *oldset, struct pt_regs *regs)</programlisting>
**/
```
It is recommended that you use the `<programlisting>` tag in this instance, since overriding the `Synopsis` content of an entry does not automatically form the necessary tags.

For the purposes of improving the DocBook XML output of your comments, you can also use the following XML tags in your comments:

- `command`
- `emphasis`
- `programlisting`
- `remark` (tagged strings will appear in Publican beta builds of the document)
CHAPTER 3. 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**
function::addr — Address of the current probe point.

**SYNOPSIS**

```
addr:long()
```

**ARGUMENTS**
None

**DESCRIPTION**
Returns the instruction pointer from the current probe's register state. Not all probe types have registers though, in which case zero is returned. The returned address is suitable for use with functions like `symname` and `symdata`.

**NAME**
function::asmlinkage — Mark function as declared asmlinkage

**SYNOPSIS**

```
asmlinkage()
```

**ARGUMENTS**
None

**DESCRIPTION**
Call this function before accessing arguments using the *_arg functions if the probed kernel function was declared asmlinkage in the source.

**NAME**
function::backtrace — Hex backtrace of current kernel stack

**SYNOPSIS**

```
backtrace:string()
```

**ARGUMENTS**
None
DESCRIPTION
This function returns a string of hex addresses that are a backtrace of the kernel stack. Output may be truncated as per maximum string length (MAXSTRINGLEN). See `ubacktrace` for user-space backtrace.

NAME
`function::caller` — Return name and address of calling function

SYNOPSIS
```
caller:string()
```

ARGUMENTS
None

DESCRIPTION
This function returns the address and name of the calling function. This is equivalent to calling: `sprintf("\0x", symname(caller_addr), caller_addr)`

NAME
`function::caller_addr` — Return caller address

SYNOPSIS
```
caller_addr:long()
```

ARGUMENTS
None

DESCRIPTION
This function returns the address of the calling function.

NAME
`function::callers` — Return first n elements of kernel stack backtrace

SYNOPSIS
```
callers:string(n:long)
```

ARGUMENTS

\( n \)

- number of levels to descend in the stack (not counting the top level). If \( n \) is -1, print the entire stack.
DESCRIPTION
This function returns a string of the first n hex addresses from the backtrace of the kernel stack. Output may be truncated as per maximum string length (MAXSTRINGLEN).

NAME
function::cmdline_arg — Fetch a command line argument

SYNOPSIS

    cmdline_arg:string(n:long)

ARGUMENTS

    n
    Argument to get (zero is the program itself)

DESCRIPTION
Returns argument the requested argument from the current process or the empty string when there are not that many arguments or there is a problem retrieving the argument. Argument zero is traditionally the command itself.

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

SYNOPSIS

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

ARGUMENTS

    n
    First argument to get (zero is normally the program itself)

    m
    Last argument to get (or minus one for all arguments after n)

    delim
    String to use to separate arguments when more than one.

DESCRIPTION
Returns arguments from the current process starting with argument number n, up to argument m. If there are less than n arguments, or the arguments cannot be retrieved from the current process, the empty string is returned. If m is smaller than n then all arguments starting from argument n are returned. Argument zero is traditionally the command itself.
NAME
function::cmdline_str — Fetch all command line arguments from current process

SYNOPSIS

```plaintext
cmdline_str:string()
```

ARGUMENTS
None

DESCRIPTION
Returns all arguments from the current process delimited by spaces. Returns the empty string when the arguments cannot be retrieved.

NAME
function::cpu — Returns the current cpu number

SYNOPSIS

```plaintext
cpu:long()
```

ARGUMENTS
None

DESCRIPTION
This function returns the current cpu number.

NAME
function::cpuid — Returns the current cpu number

SYNOPSIS

```plaintext
cpuid:long()
```

ARGUMENTS
None

DESCRIPTION
This function returns the current cpu number. Deprecated in SystemTap 1.4 and removed in SystemTap 1.5.
NAME
function::egid — Returns the effective gid of a target process

SYNOPSIS

| egid:long() |

ARGUMENTS

None

DESCRIPTION

This function returns the effective gid of a target process.

NAME
function::env_var — Fetch environment variable from current process

SYNOPSIS

| env_var:string(name:string) |

ARGUMENTS

name

Name of the environment variable to fetch

DESCRIPTION

Returns the contents of the specified environment value for the current process. If the variable isn't set an empty string is returned.

NAME
function::euid — Return the effective uid of a target process

SYNOPSIS

| euid:long() |

ARGUMENTS

None

DESCRIPTION

Returns the effective user ID of the target process.
**function::execname — Returns the execname of a target process (or group of processes)**

**SYNOPSIS**

```plaintext
execname: string()
```

**ARGUMENTS**

None

**DESCRIPTION**

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

---

**NAME**

**function::fastcall — Mark function as declared fastcall**

**SYNOPSIS**

```plaintext
fastcall()
```

**ARGUMENTS**

None

**DESCRIPTION**

Call this function before accessing arguments using the *arg functions if the probed kernel function was declared fastcall in the source.

---

**NAME**

**function::gid — Returns the group ID of a target process**

**SYNOPSIS**

```plaintext
gid: long()
```

**ARGUMENTS**

None

**DESCRIPTION**

This function returns the group ID of a target process.

---

**NAME**

**function::int_arg — Return function argument as signed int**

**SYNOPSIS**

```plaintext
```

---
ARGUMENTS
n
index of argument to return

DESCRIPTION
Return the value of argument n as a signed int (i.e., a 32-bit integer sign-extended to 64 bits).

NAME
function::is_myproc — Determines if the current probe point has occurred in the user's own process

SYNOPSIS
is_myproc:long()

ARGUMENTS
None

DESCRIPTION
This function returns 1 if the current probe point has occurred in the user's own process.

NAME
function::is_return — Whether the current probe context is a return probe

SYNOPSIS
is_return:long()

ARGUMENTS
None

DESCRIPTION
Returns 1 if the current probe context is a return probe, returns 0 otherwise.

NAME
function::long_arg — Return function argument as signed long

SYNOPSIS
long_arg:long(n:long)
ARGUMENTS

\( n \)

index of argument to return

DESCRIPTION

Return the value of argument \( n \) as a signed long. On architectures where a long is 32 bits, the value is sign-extended to 64 bits.

NAME

function::longlong_arg — Return function argument as 64-bit value

SYNOPSIS

| longlong_arg:long(n:long) |

ARGUMENTS

\( n \)

index of argument to return

DESCRIPTION

Return the value of argument \( n \) as a 64-bit value.

NAME

function::modname — Return the kernel module name loaded at the address

SYNOPSIS

| modname:string(addr:long) |

ARGUMENTS

\( addr \)

The address to map to a kernel module name

DESCRIPTION

Returns the module name associated with the given address if known. If not known it will raise an error. If the address was not in a kernel module, but in the kernel itself, then the string “kernel” will be returned.
function::module_name — The module name of the current script

SYNOPSIS

module_name:string()

ARGUMENTS
None

DESCRIPTION
This function returns the name of the stap module. Either generated randomly (stap_[0-9a-f]+_[0-9a-f]+) or set by stap -m <module_name>.

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

SYNOPSIS

module_size:string()

ARGUMENTS
None

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

NAME
function::ns_egid — Returns the effective gid of a target process as seen in a user namespace

SYNOPSIS

ns_egid:long()

ARGUMENTS
None

DESCRIPTION
This function returns the effective gid of a target process as seen in the target user namespace if provided, or the stap process namespace.

NAME
function::ns_euid — Returns the effective user ID of a target process as seen in a user namespace

SYNOPSIS
ARGUMENTS
None

DESCRIPTION
This function returns the effective user ID of the target process as seen in the target user namespace if provided, or the stap process namespace.

NAME
function::ns_euid — Returns the group ID of a target process as seen in a user namespace

SYNOPSIS

ns_euid:long()

ARGUMENTS
None

DESCRIPTION
This function returns the group ID of a target process as seen in the target user namespace if provided, or the stap process namespace.

NAME
function::ns_gid — Returns the process group ID of the current process as seen in a pid namespace

SYNOPSIS

ns_gid:long()

ARGUMENTS
None

DESCRIPTION
This function returns the process group ID of the current process as seen in the target pid namespace if provided, or the stap process namespace.

NAME
function::ns_pgrp — Returns the ID of a target process as seen in a pid namespace

SYNOPSIS

ns_pgrp:long()

ARGUMENTS
None

DESCRIPTION
This function returns the ID of a target process as seen in the target user namespace if provided, or the stap process namespace.
ARGUMENTS
None

DESCRIPTION
This function returns the ID of a target process as seen in the target pid namespace.

NAME
function::ns_ppid — Returns the process ID of a target process's parent process as seen in a pid namespace

SYNOPSIS

ARGUMENTS
None

DESCRIPTION
This function returns the process ID of the target process's parent process as seen in the target pid namespace if provided, or the stap process namespace.

NAME
function::ns_sid — Returns the session ID of the current process as seen in a pid namespace

SYNOPSIS

ARGUMENTS
None

DESCRIPTION
The namespace-aware session ID of a process is the process group ID of the session leader as seen in the target pid namespace if provided, or the stap process namespace. Session ID is stored in the signal_struct since Kernel 2.6.0.

NAME
function::ns_tid — Returns the thread ID of a target process as seen in a pid namespace

SYNOPSIS

ARGUMENTS
DESCRIPTION
This function returns the thread ID of a target process as seen in the target pid namespace if provided, or the stap process namespace.

NAME
function::ns_uid — Returns the user ID of a target process as seen in a user namespace

SYNOPSIS
\[ \text{ns_uid}: \text{long}() \]

ARGUMENTS
None

DESCRIPTION
This function returns the user ID of the target process as seen in the target user namespace if provided, or the stap process namespace.

NAME
function::pexecname — Returns the execname of a target process's parent process

SYNOPSIS
\[ \text{pexecname}: \text{string}() \]

ARGUMENTS
None

DESCRIPTION
This function returns the execname of a target process's parent process.

NAME
function::pgrp — Returns the process group ID of the current process

SYNOPSIS
\[ \text{pgrp}: \text{long}() \]

ARGUMENTS
None

DESCRIPTION
This function returns the process group ID of the current process.

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

**SYNOPSIS**

`pid:long()`

**ARGUMENTS**
None

**DESCRIPTION**
This function returns the ID of a target process.

**NAME**
function::pid2execname — The name of the given process identifier

**SYNOPSIS**

`pid2execname:string(pid:long)`

**ARGUMENTS**

`pid`

  process identifier

**DESCRIPTION**
Return the name of the given process id.

**NAME**
function::pid2task — The task_struct of the given process identifier

**SYNOPSIS**

`pid2task:long(pid:long)`

**ARGUMENTS**

`pid`

  process identifier
DESCRIPTION
Return the task struct of the given process id.

NAME
function::pn — Returns the active probe name

SYNOPSIS
pn:string()

ARGUMENTS
None

DESCRIPTION
This function returns the script-level probe point associated with a currently running probe handler, including wild-card expansion effects. Context: The current probe point.

NAME
function::pnlabel — Returns the label name parsed from the probe name

SYNOPSIS
pnlabel:string()

ARGUMENTS
None

DESCRIPTION
This returns the label name as parsed from the script-level probe point. This function will only work if called directly from the body of a '.label' probe point (i.e. no aliases).

CONTEXT
The current probe point.

NAME
function::pointer_arg — Return function argument as pointer value

SYNOPSIS
pointer_arg:long(n:long)

ARGUMENTS
n
index of argument to return

**DESCRIPTION**
Return the unsigned value of argument \( n \), same as \texttt{ulong\_arg}. Can be used with any type of pointer.

**NAME**
function::\texttt{pp} — Returns the active probe point

**SYNOPSIS**

```
\texttt{pp}: \texttt{string()}
```

**ARGUMENTS**
None

**DESCRIPTION**
This function returns the fully-resolved probe point associated with a currently running probe handler, including alias and wild-card expansion effects. Context: The current probe point.

**NAME**
function::\texttt{ppfunc} — Returns the function name parsed from \texttt{pp}

**SYNOPSIS**

```
\texttt{ppfunc}: \texttt{string()}
```

**ARGUMENTS**
None

**DESCRIPTION**
This returns the function name from the current \texttt{pp}. Not all \texttt{pp} have functions in them, in which case """" is returned.

**NAME**
function::\texttt{ppid} — Returns the process ID of a target process's parent process

**SYNOPSIS**

```
\texttt{ppid}: \texttt{long()}
```

**ARGUMENTS**
None
DESCRIPTION
This function return the process ID of the target process's parent process.

NAME
function::print_backtrace — Print kernel stack back trace

SYNOPSIS

```
print_backtrace()
```

ARGUMENTS
None

DESCRIPTION
This function is equivalent to print_stack(backtrace), except that deeper stack nesting may be supported. See print_ubacktrace for user-space backtrace. The function does not return a value.

NAME
function::print_regs — Print a register dump

SYNOPSIS

```
print_regs()
```

ARGUMENTS
None

DESCRIPTION
This function prints a register dump. Does nothing if no registers are available for the probe point.

NAME
function::print_stack — Print out kernel stack from string

SYNOPSIS

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

ARGUMENTS

```
stk
```
String with list of hexadecimal addresses

DESCRIPTION
This function performs a symbolic lookup of the addresses in the given string, which is assumed to be the result of a prior call to `backtrace`.

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

**NOTE**

It is recommended to use `print_syms` instead of this function.

---

**NAME**

`function::print_syms` — Print out kernel stack from string

**SYNOPSIS**

```
print_syms(callers:string)
```

**ARGUMENTS**

`callers`

String with list of hexadecimal (kernel) addresses

**DESCRIPTION**

This function performs a symbolic lookup of the addresses in the given string, which are assumed to be the result of prior calls to `stack`, `callers`, and similar functions.

Prints one line per address, including the address, the name of the function containing the address, and an estimate of its position within that function, as obtained by `symdata`. Returns nothing.

---

**NAME**

`function::print_ubacktrace` — Print stack back trace for current user-space task.

**SYNOPSIS**

```
print_ubacktrace()
```

**ARGUMENTS**

None

**DESCRIPTION**

Equivalent to `print_ustack(ubacktrace)`, except that deeper stack nesting may be supported. Returns nothing. See `print_backtrace` for kernel backtrace.

**NOTE**

To get (full) backtraces for user space applications and shared shared libraries not mentioned in the current script run stap with `-d /path/to/exe-or-so` and/or add `--ldd` to load all needed unwind data.
NAME
function::print_ubacktrace_brief — Print stack back trace for current user-space task.

SYNOPSIS

print_ubacktrace_brief()

ARGUMENTS
None

DESCRIPTION
Equivalent to print_ubacktrace, but output for each symbol is shorter (just name and offset, or just
the hex address of no symbol could be found).

NOTE
To get (full) backtraces for user space applications and shared shared libraries not mentioned in the
current script run stap with -d /path/to/exe-or-so and/or add --ldd to load all needed unwind data.

NAME
function::print_ustack — Print out stack for the current task from string.

SYNOPSIS

print_ustack(stk:string)

ARGUMENTS

stk
String with list of hexadecimal addresses for the current task.

DESCRIPTION
Perform a symbolic lookup of the addresses in the given string, which is assumed to be the result of a
prior call to ubacktrace for the current task.

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

NOTE
it is recommended to use print_usyms instead of this function.

NAME
function::print_usyms — Print out user stack from string

SYNOPSIS

print_usyms(callers:string)
ARGUMENTS

callers
String with list of hexadecimal (user) addresses

DESCRIPTION
This function performs a symbolic lookup of the addresses in the given string, which are assumed to be the result of prior calls to `ustack`, `ucallers`, and similar functions.

Prints one line per address, including the address, the name of the function containing the address, and an estimate of its position within that function, as obtained by `usymdata`. Returns nothing.

NAME
function::probe_type — The low level probe handler type of the current probe.

SYNOPSIS

ARGUMENTS
None

DESCRIPTION
Returns a short string describing the low level probe handler type for the current probe point. This is for informational purposes only. Depending on the low level probe handler different context functions can or cannot provide information about the current event (for example some probe handlers only trigger in user space and have no associated kernel context). High-level probes might map to the same or different low-level probes (depending on systemtap version and/or kernel used).

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

SYNOPSIS

ARGUMENTS
None

DESCRIPTION
This function returns the name of the function being probed based on the current address, as computed by `symname(addr)` or `usymname(uaddr)` depending on probe context (whether the probe is a user probe or a kernel probe).

PLEASE NOTE
this function’s behaviour differs between SystemTap 2.0 and earlier versions. Prior to 2.0, `probefunc` obtained the function name from the probe point string as returned by `pp`, and used the current address as a fallback.

Consider using `ppfunc` instead.

---

**NAME**

function::probemod — Return the probe point’s kernel module name

**SYNOPSIS**

```plaintext
probemod:string()
```

**ARGUMENTS**

None

**DESCRIPTION**

This function returns the name of the kernel module containing the probe point, if known.

---

**NAME**

function::pstrace — Chain of processes and pids back to init(1)

**SYNOPSIS**

```plaintext
pstrace:string(task:long)
```

**ARGUMENTS**

`task`

Pointer to task struct of process

**DESCRIPTION**

This function returns a string listing execname and pid for each process starting from `task` back to the process ancestor that init(1) spawned.

---

**NAME**

function::register — Return the signed value of the named CPU register

**SYNOPSIS**

```plaintext
register:long(name:string)
```

**ARGUMENTS**
**NAME**

function::registers_valid — Determines validity of **register** and **u_register** in current context

**SYNOPSIS**

```c
registers_valid:long()
```

**ARGUMENTS**

None

**DESCRIPTION**

This function returns 1 if **register** and **u_register** can be used in the current context, or 0 otherwise. For example, **registers_valid** returns 0 when called from a begin or end probe.

---

**NAME**

function::regparm — Specify regparm value used to compile function

**SYNOPSIS**

```c
regparm(n:long)
```

**ARGUMENTS**

- **n**
original regparm value

DESCRIPTION
Call this function with argument n before accessing function arguments using the *_arg function is the function was build with the gcc -mregparm=n option.

(The i386 kernel is built with \-mregparm=3, so systemtap considers regparm(3) the default for kernel functions on that architecture.) Only valid on i386 and x86_64 (when probing 32bit applications). Produces an error on other architectures.

NAME
function::remote_id — The index of this instance in a remote execution.

SYNOPSIS
remote_id:long()

ARGUMENTS
None

DESCRIPTION
This function returns a number 0..N, which is the unique index of this particular script execution from a swarm of “stap --remote A --remote B ...” runs, and is the same number “stap --remote-prefix” would print. The function returns -1 if the script was not launched with “stap --remote”, or if the remote staprun/stapsh are older than version 1.7.

NAME
function::remote_uri — The name of this instance in a remote execution.

SYNOPSIS
remote_uri:string()

ARGUMENTS
None

DESCRIPTION
This function returns the remote host used to invoke this particular script execution from a swarm of “stap --remote” runs. It may not be unique among the swarm. The function returns an empty string if the script was not launched with “stap --remote”.

NAME
function::s32_arg — Return function argument as signed 32-bit value
SYNOPSIS

s32_arg::long(n:long)

ARGUMENTS

n
index of argument to return

DESCRIPTION
Return the signed 32-bit value of argument n, same as int_arg.

NAME
function::s64_arg — Return function argument as signed 64-bit value

SYNOPSIS

s64_arg::long(n:long)

ARGUMENTS

n
index of argument to return

DESCRIPTION
Return the signed 64-bit value of argument n, same as longlong_arg.

NAME
function::sid — Returns the session ID of the current process

SYNOPSIS

sid:long()

ARGUMENTS

None

DESCRIPTION
The session ID of a process is the process group ID of the session leader. Session ID is stored in the signal_struct since Kernel 2.6.0.

NAME
function::sprint_backtrace — Return stack back trace as string

SYNOPSIS

sprint_backtrace:string()

ARGUMENTS

None

DESCRIPTION

Returns a simple (kernel) backtrace. One line per address. Includes the symbol name (or hex address if symbol couldn't be resolved) and module name (if found). Includes the offset from the start of the function if found, otherwise the offset will be added to the module (if found, between brackets). Returns the backtrace as string (each line terminated by a newline character). Note that the returned stack will be truncated to MAXSTRINGLEN, to print fuller and richer stacks use print_backtrace. Equivalent to sprint_stack(backtrace), but more efficient (no need to translate between hex strings and final backtrace string).

NAME

function::sprint_stack — Return stack for kernel addresses from string

SYNOPSIS

sprint_stack:string(stk:string)

ARGUMENTS

stk

String with list of hexadecimal (kernel) addresses

DESCRIPTION

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

Returns a simple backtrace from the given hex string. One line per address. Includes the symbol name (or hex address if symbol couldn't be resolved) and module name (if found). Includes the offset from the start of the function if found, otherwise the offset will be added to the module (if found, between brackets). Returns the backtrace as string (each line terminated by a newline character). Note that the returned stack will be truncated to MAXSTRINGLEN, to print fuller and richer stacks use print_stack.

NOTE

It is recommended to use sprint_syms instead of this function.

NAME

function::sprint_syms — Return stack for kernel addresses from string
SYNOPSIS

sprint_syms(callers:string)

ARGUMENTS

callers

String with list of hexadecimal (kernel) addresses

DESCRIPTION

Perform a symbolic lookup of the addresses in the given string, which are assumed to be the result of a prior calls to stack, callers, and similar functions.

Returns a simple backtrace from the given hex string. One line per address. Includes the symbol name (or hex address if symbol couldn’t be resolved) and module name (if found), as obtained from symdata. Includes the offset from the start of the function if found, otherwise the offset will be added to the module (if found, between brackets). Returns the backtrace as string (each line terminated by a newline character). Note that the returned stack will be truncated to MAXSTRINGLEN, to print fuller and richer stacks use print_syms.

NAME

function::sprint_ubacktrace — Return stack back trace for current user-space task as string.

SYNOPSIS

sprint_ubacktrace:string()

ARGUMENTS

None

DESCRIPTION

Returns a simple backtrace for the current task. One line per address. Includes the symbol name (or hex address if symbol couldn’t be resolved) and module name (if found). Includes the offset from the start of the function if found, otherwise the offset will be added to the module (if found, between brackets). Returns the backtrace as string (each line terminated by a newline character). Note that the returned stack will be truncated to MAXSTRINGLEN, to print fuller and richer stacks use print_ubacktrace. Equivalent to sprint_ustack(ubacktrace), but more efficient (no need to translate between hex strings and final backtrace string).

NOTE

To get (full) backtraces for user space applications and shared shared libraries not mentioned in the current script run stap with -d /path/to/exe-or-so and/or add --ldd to load all needed unwind data.

NAME

function::sprint_ustack — Return stack for the current task from string.
SYNOPSIS

| function: string(stk:string) |

ARGUMENTS

stk

String with list of hexadecimal addresses for the current task.

DESCRIPTION

Perform a symbolic lookup of the addresses in the given string, which is assumed to be the result of a prior call to ubacktrace for the current task.

Returns a simple backtrace from the given hex string. One line per address. Includes the symbol name (or hex address if symbol couldn’t be resolved) and module name (if found). Includes the offset from the start of the function if found, otherwise the offset will be added to the module (if found, between brackets). Returns the backtrace as string (each line terminated by a newline character). Note that the returned stack will be truncated to MAXSTRINGLEN, to print fuller and richer stacks use print_ustack.

NOTE

It is recommended to use sprint_usyms instead of this function.

NAME

function::sprint_usyms — Return stack for user addresses from string

SYNOPSIS

| function: sprint_usyms(callers:string) |

ARGUMENTS

callers

String with list of hexadecimal (user) addresses

DESCRIPTION

Perform a symbolic lookup of the addresses in the given string, which are assumed to be the result of a prior calls to ustack, ucallers, and similar functions.

Returns a simple backtrace from the given hex string. One line per address. Includes the symbol name (or hex address if symbol couldn’t be resolved) and module name (if found), as obtained from usymdata. Includes the offset from the start of the function if found, otherwise the offset will be added to the module (if found, between brackets). Returns the backtrace as string (each line terminated by a newline character). Note that the returned stack will be truncated to MAXSTRINGLEN, to print fuller and richer stacks use print_usyms.
NAME  
function::stack — Return address at given depth of kernel stack backtrace

SYNOPSIS

stack:long(n:long)

ARGUMENTS

n

number of levels to descend in the stack.

DESCRIPTION

Performs a simple (kernel) backtrace, and returns the element at the specified position. The results of the backtrace itself are cached, so that the backtrace computation is performed at most once no matter how many times stack is called, or in what order.

NAME  
function::stack_size — Return the size of the kernel stack

SYNOPSIS

stack_size:long()

ARGUMENTS

None

DESCRIPTION

This function returns the size of the kernel stack.

NAME  
function::stack_unused — Returns the amount of kernel stack currently available

SYNOPSIS

stack_unused:long()

ARGUMENTS

None

DESCRIPTION

This function determines how many bytes are currently available in the kernel stack.
NAME
function::stack_used — Returns the amount of kernel stack used

SYNOPSIS

```
stack_used:long()
```

ARGUMENTS

None

DESCRIPTION

This function determines how many bytes are currently used in the kernel stack.

NAME
function::stp_pid — The process id of the stapio process

SYNOPSIS

```
stp_pid:long()
```

ARGUMENTS

None

DESCRIPTION

This function returns the process id of the stapio process that launched this script. There could be other SystemTap scripts and stapio processes running on the system.

NAME
function::symdata — Return the kernel symbol and module offset for the address

SYNOPSIS

```
symdata:string(addr:long)
```

ARGUMENTS

```
addr
```

The address to translate

DESCRIPTION

Returns the (function) symbol name associated with the given address if known, the offset from the start and size of the symbol, plus module name (between brackets). If symbol is unknown, but module is known, the offset inside the module, plus the size of the module is added. If any element is not known it will be omitted and if the symbol name is unknown it will return the hex string for the given address.
NAME
function::symfile — Return the file name of a given address.

SYNOPSIS

   symfile:string(addr:long)

ARGUMENTS

   addr
       The address to translate.

DESCRIPTION
Returns the file name of the given address, if known. If the file name cannot be found, the hex string representation of the address will be returned.

NAME
function::symfileline — Return the file name and line number of an address.

SYNOPSIS

   symfileline:string(addr:long)

ARGUMENTS

   addr
       The address to translate.

DESCRIPTION
Returns the file name and the (approximate) line number of the given address, if known. If the file name or the line number cannot be found, the hex string representation of the address will be returned.

NAME
function::symline — Return the line number of an address.

SYNOPSIS

   symline:string(addr:long)

ARGUMENTS

   addr
       The address to translate.
DESCRIPTION
Returns the (approximate) line number of the given address, if known. If the line number cannot be found, the hex string representation of the address will be returned.

NAME
function::symname — Return the kernel symbol associated with the given address

SYNOPSIS

    symname:string(addr:long)

ARGUMENTS

addr
    The address to translate

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

NAME
function::target — Return the process ID of the target process

SYNOPSIS

    target:long()

ARGUMENTS
None

DESCRIPTION
This function returns the process ID of the target process. This is useful in conjunction with the -x PID or -c CMD command-line options to stap. An example of its use is to create scripts that filter on a specific process.

-x <pid> target returns the pid specified by -x

target returns the pid for the executed command specified by -c

NAME
function::task_ancestry — The ancestry of the given task

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

ARGUMENTS

*task*

  task_struct pointer

*with_time*

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

DESCRIPTION

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

NAME

function::task_backtrace — Hex backtrace of an arbitrary task

SYNOPSIS

  task_backtrace::string(task:long)

ARGUMENTS

*task*

  pointer to task_struct

DESCRIPTION

This function returns a string of hex addresses that are a backtrace of the stack of a particular task. Output may be truncated as per maximum string length. Deprecated in SystemTap 1.6.

NAME

function::task_cpu — The scheduled cpu of the task

SYNOPSIS

  task_cpu::long(task:long)

ARGUMENTS

*task*

  task_struct pointer

DESCRIPTION

This function returns the scheduled cpu for the given task.
NAME
function::task_current — The current task_struct of the current task

SYNOPSIS

task_current:long()

ARGUMENTS
None

DESCRIPTION
This function returns the task_struct representing the current process. This address can be passed to the various task_*() functions to extract more task-specific data.

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

SYNOPSIS

task_cwd_path:long(task:long)

ARGUMENTS

  task
    task_struct pointer.

NAME
function::task_egid — The effective group identifier of the task

SYNOPSIS

task_egid:long(task:long)

ARGUMENTS

  task
    task_struct pointer

DESCRIPTION
This function returns the effective group id of the given task.
NAME
function::task_euid — The effective user identifier of the task

SYNOPSIS

```
    task_euid:long(task:long)
```

ARGUMENTS

```
task
    task_struct pointer
```

DESCRIPTION

This function returns the effective user id of the given task.

NAME

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

SYNOPSIS

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

ARGUMENTS

```
task
    task_struct pointer.
```

NAME

function::task_execname — The name of the task

SYNOPSIS

```
    task_execname:string(task:long)
```

ARGUMENTS

```
task
    task_struct pointer
```

DESCRIPTION

Return the name of the given task.
NAME
function::task_fd_lookup — get the file struct for a task's fd

SYNOPSIS

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

ARGUMENTS

- `task`: task_struct pointer.
- `fd`: file descriptor number.

DESCRIPTION

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

NAME
function::task_gid — The group identifier of the task

SYNOPSIS

```
task_gid:long(task:long)
```

ARGUMENTS

- `task`: task_struct pointer

DESCRIPTION

This function returns the group id of the given task.

NAME
function::task_max_file_handles — The max number of open files for the task

SYNOPSIS

```
task_max_file_handles:long(task:long)
```

ARGUMENTS

- `task`: task_struct pointer
**DESCRIPTION**

This function returns the maximum number of file handlers for the given task.

**NAME**

function::task_nice — The nice value of the task

**SYNOPSIS**

```
task_nice:long(task:long)
```

**ARGUMENTS**

`task`

task_struct pointer

**DESCRIPTION**

This function returns the nice value of the given task.

**NAME**

function::task_ns_egid — The effective group identifier of the task

**SYNOPSIS**

```
task_ns_egid:long(task:long)
```

**ARGUMENTS**

`task`

task_struct pointer

**DESCRIPTION**

This function returns the effective group id of the given task.

**NAME**

function::task_ns_euid — The effective user identifier of the task

**SYNOPSIS**

```
task_ns_euid:long(task:long)
```

**ARGUMENTS**

`task`


DESCRIPTION
This function returns the effective user id of the given task.

NAME
function::task_ns_gid — The group identifier of the task as seen in a namespace

SYNOPSIS
  task_ns_gid:long(task:long)

ARGUMENTS
  task
  task_struct pointer

DESCRIPTION
This function returns the group id of the given task as seen in the given user namespace.

NAME
function::task_ns_pid — The process identifier of the task

SYNOPSIS
  task_ns_pid:long(task:long)

ARGUMENTS
  task
  task_struct pointer

DESCRIPTION
This function returns the process id of the given task based on the specified pid namespace.

NAME
function::task_ns_tid — The thread identifier of the task as seen in a namespace

SYNOPSIS
  task_ns_tid:long(task:long)
ARGUMENTS

*task*

  task_struct pointer

DESCRIPTION
This function returns the thread id of the given task as seen in the pid namespace.

NAME
function::task_ns_uid — The user identifier of the task

SYNOPSIS

  task_ns_uid:long(task:long)

ARGUMENTS

*task*

  task_struct pointer

DESCRIPTION
This function returns the user id of the given task.

NAME
function::task_open_file_handles — The number of open files of the task

SYNOPSIS

  task_open_file_handles:long(task:long)

ARGUMENTS

*task*

  task_struct pointer

DESCRIPTION
This function returns the number of open file handlers for the given task.

NAME
function::task_parent — The task_struct of the parent task
SYNOPSIS

```c
task_parent:long(task:long)
```

ARGUMENTS

`task`

`task_struct` pointer

DESCRIPTION

This function returns the parent `task_struct` of the given task. This address can be passed to the various task_*() functions to extract more task-specific data.

NAME

function::task_pid — The process identifier of the task

SYNOPSIS

```c
task_pid:long(task:long)
```

ARGUMENTS

`task`

`task_struct` pointer

DESCRIPTION

This function returns the process id of the given task.

NAME

function::task_prio — The priority value of the task

SYNOPSIS

```c
task_prio:long(task:long)
```

ARGUMENTS

`task`

`task_struct` pointer

DESCRIPTION

This function returns the priority value of the given task.
NAME
function::task_state — The state of the task

SYNOPSIS

\[ \text{task}\_\text{state}: \text{long}(\text{task}: \text{long}) \]

ARGUMENTS

\textit{task}

- task_struct pointer

DESCRIPTION

Return the state of the given task, one of: TASK_RUNNING (0), TASK_INTERRUPTIBLE (1), TASK_UNINTERRUPTIBLE (2), TASK_STOPPED (4), TASK_TRACED (8), EXIT_ZOMBIE (16), or EXIT_DEAD (32).

NAME
function::task_tid — The thread identifier of the task

SYNOPSIS

\[ \text{task}\_\text{tid}: \text{long}(\text{task}: \text{long}) \]

ARGUMENTS

\textit{task}

- task_struct pointer

DESCRIPTION

This function returns the thread id of the given task.

NAME
function::task_uid — The user identifier of the task

SYNOPSIS

\[ \text{task}\_\text{uid}: \text{long}(\text{task}: \text{long}) \]

ARGUMENTS

\textit{task}

- task_struct pointer
**DESCRIPTION**
This function returns the user id of the given task.

---

**NAME**
function::tid — Returns the thread ID of a target process

**SYNOPSIS**
```plaintext
tid:long()
```

**ARGUMENTS**
None

**DESCRIPTION**
This function returns the thread ID of the target process.

---

**NAME**
function::u32_arg — Return function argument as unsigned 32-bit value

**SYNOPSIS**
```plaintext
u32_arg:long(n:long)
```

**ARGUMENTS**

\( n \)

- index of argument to return

**DESCRIPTION**
Return the unsigned 32-bit value of argument \( n \), same as uint_arg.

---

**NAME**
function::u64_arg — Return function argument as unsigned 64-bit value

**SYNOPSIS**
```plaintext
u64_arg:long(n:long)
```

**ARGUMENTS**

\( n \)

- index of argument to return
DESCRIPTION
Return the unsigned 64-bit value of argument n, same as ulonglong_arg.

NAME
function::u_register — Return the unsigned value of the named CPU register

SYNOPSIS
u_register:long(name:string)

ARGUMENTS
name
Name of the register to return

DESCRIPTION
Same as register(name), except that if the register is 32 bits wide, it is zero-extended to 64 bits.

NAME
function::uaddr — User space address of current running task

SYNOPSIS
uaddr:long()

ARGUMENTS
None

DESCRIPTION
Returns the address in userspace that the current task was at when the probe occurred. When the current running task isn't a user space thread, or the address cannot be found, zero is returned. Can be used to see where the current task is combined with usymname or usymdata. Often the task will be in the VDSO where it entered the kernel.

NAME
function::ubacktrace — Hex backtrace of current user-space task stack.

SYNOPSIS
ubacktrace:string()

ARGUMENTS
None
DESCRIPTION
Return a string of hex addresses that are a backtrace of the stack of the current task. Output may be truncated as per maximum string length. Returns empty string when current probe point cannot determine user backtrace. See backtrace for kernel traceback.

NOTE
To get (full) backtraces for user space applications and shared shared libraries not mentioned in the current script run stap with -d /path/to/exe-or-so and/or add --ldd to load all needed unwind data.

NAME
function::ucallers — Return first n elements of user stack backtrace

SYNOPSIS
  ucallers:string(n:long)

ARGUMENTS
  n
  number of levels to descend in the stack (not counting the top level). If n is -1, print the entire stack.

DESCRIPTION
This function returns a string of the first n hex addresses from the backtrace of the user stack. Output may be truncated as per maximum string length (MAXSTRINGLEN).

NOTE
To get (full) backtraces for user space applications and shared shared libraries not mentioned in the current script run stap with -d /path/to/exe-or-so and/or add --ldd to load all needed unwind data.

NAME
function::uid — Returns the user ID of a target process

SYNOPSIS
  uid:long()

ARGUMENTS
  None

DESCRIPTION
This function returns the user ID of the target process.
function::uint_arg — Return function argument as unsigned int

SYNOPSIS

uint_arg::long(n:long)

ARGUMENTS

n

index of argument to return

DESCRIPTION

Return the value of argument n as an unsigned int (i.e., a 32-bit integer zero-extended to 64 bits).

NAME

function::ulong_arg — Return function argument as unsigned long

SYNOPSIS

ulong_arg::long(n:long)

ARGUMENTS

n

index of argument to return

DESCRIPTION

Return the value of argument n as an unsigned long. On architectures where a long is 32 bits, the value is zero-extended to 64 bits.

NAME

function::ulonglong_arg — Return function argument as 64-bit value

SYNOPSIS

ulonglong_arg::long(n:long)

ARGUMENTS

n

index of argument to return

DESCRIPTION
Return the value of argument \( n \) as a 64-bit value. (Same as `longlong_arg`.)

**NAME**
function::umodname — Returns the (short) name of the user module.

**SYNOPSIS**
```
 umodname:string(addr:long)
```

**ARGUMENTS**

\( addr \)

User-space address

**DESCRIPTION**
Returns the short name of the user space module for the current task that that the given address is part of. Reports an error when the address isn't in a (mapped in) module, or the module cannot be found for some reason.

**NAME**
function::user_mode — Determines if probe point occurs in user-mode

**SYNOPSIS**
```
 user_mode:long()
```

**ARGUMENTS**
None

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

**NAME**
function::ustack — Return address at given depth of user stack backtrace

**SYNOPSIS**
```
 ustack:long(n:long)
```

**ARGUMENTS**

\( n \)

number of levels to descend in the stack.
DESCRIPTION
Performs a simple (user space) backtrace, and returns the element at the specified position. The results of the backtrace itself are cached, so that the backtrace computation is performed at most once no matter how many times usack is called, or in what order.

NAME
function::usymdata — Return the symbol and module offset of an address.

SYNOPSIS
usymdata:string(addr:long)

ARGUMENTS
addr
The address to translate.

DESCRIPTION
Returns the (function) symbol name associated with the given address in the current task if known, the offset from the start and the size of the symbol, plus the module name (between brackets). If symbol is unknown, but module is known, the offset inside the module, plus the size of the module is added. If any element is not known it will be omitted and if the symbol name is unknown it will return the hex string for the given address.

NAME
function::usymfile — Return the file name of a given address.

SYNOPSIS
usymfile:string(addr:long)

ARGUMENTS
addr
The address to translate.

DESCRIPTION
Returns the file name of the given address, if known. If the file name cannot be found, the hex string representation of the address will be returned.

NAME
function::usymfileline — Return the file name and line number of an address.
SYNOPSIS
usymfileline: string(addr: long)

ARGUMENTS

addr
The address to translate.

DESCRIPTION
Returns the file name and the (approximate) line number of the given address, if known. If the file name
or the line number cannot be found, the hex string representation of the address will be returned.

NAME
function::usymfileline — Return the line number of an address.

SYNOPSIS
usymfileline: string(addr: long)

ARGUMENTS

addr
The address to translate.

DESCRIPTION
Returns the (approximate) line number of the given address, if known. If the line number cannot be
found, the hex string representation of the address will be returned.

NAME
function::usymname — Return the symbol of an address in the current task.

SYNOPSIS
usymname: string(addr: long)

ARGUMENTS

addr
The address to translate.

DESCRIPTION
Returns the (function) symbol name associated with the given address if known. If not known it will return the hex string representation of addr.
CHAPTER 4. 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
function::HZ — Kernel HZ

SYNOPSIS
HZ:long()

ARGUMENTS
None

DESCRIPTION
This function returns the value of the kernel HZ macro, which corresponds to the rate of increase of the jiffies value.

NAME
function::cpu_clock_ms — Number of milliseconds on the given cpu's clock

SYNOPSIS
cpu_clock_ms:long(cpu:long)

ARGUMENTS
cpu
Which processor's clock to read

DESCRIPTION
This function returns the number of milliseconds on the given cpu's clock. This is always monotonic comparing on the same cpu, but may have some drift between cpus (within about a jiffy).

NAME
function::cpu_clock_ns — Number of nanoseconds on the given cpu's clock

SYNOPSIS
cpu_clock_ns:long(cpu:long)

ARGUMENTS
**NAME**

function::cpu_clock_s — Number of seconds on the given cpu's clock

**SYNOPSIS**

```
cpu_clock_s:long(cpu:long)
```

**ARGUMENTS**

`cpu`

Which processor's clock to read

**DESCRIPTION**

This function returns the number of seconds on the given cpu's clock. This is always monotonic comparing on the same cpu, but may have some drift between cpus (within about a jiffy).

---

**NAME**

function::cpu_clock_us — Number of microseconds on the given cpu's clock

**SYNOPSIS**

```
cpu_clock_us:long(cpu:long)
```

**ARGUMENTS**

`cpu`

Which processor's clock to read

**DESCRIPTION**

This function returns the number of microseconds on the given cpu's clock. This is always monotonic comparing on the same cpu, but may have some drift between cpus (within about a jiffy).

---

**NAME**

function::delete_stopwatch — Remove an existing stopwatch
**SYNOPSIS**

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

**ARGUMENTS**

`name`

the stopwatch name

**DESCRIPTION**

Remove stopwatch `name`.

**NAME**

`function::get_cycles — Processor cycle count`

**SYNOPSIS**

```c
get_cycles:long()
```

**ARGUMENTS**

None

**DESCRIPTION**

This function returns the processor cycle counter value if available, else it returns zero. The cycle counter is free running and unsynchronized on each processor. Thus, the order of events cannot determined by comparing the results of the `get_cycles` function on different processors.

**NAME**

`function::gettimeofday_ms — Number of milliseconds since UNIX epoch`

**SYNOPSIS**

```c
gettimeofday_ms:long()
```

**ARGUMENTS**

None

**DESCRIPTION**

This function returns the number of milliseconds since the UNIX epoch.

**NAME**

`function::gettimeofday_ns — Number of nanoseconds since UNIX epoch`
SYNOPSIS

gettimeofday_ns:long()

ARGUMENTS

None

DESCRIPTION

This function returns the number of nanoseconds since the UNIX epoch.

NAME

function::gettimeofday_s — Number of seconds since UNIX epoch

SYNOPSIS

gettimeofday_s:long()

ARGUMENTS

None

DESCRIPTION

This function returns the number of seconds since the UNIX epoch.

NAME

function::gettimeofday_us — Number of microseconds since UNIX epoch

SYNOPSIS

gettimeofday_us:long()

ARGUMENTS

None

DESCRIPTION

This function returns the number of microseconds since the UNIX epoch.

NAME

function::jiffies — Kernel jiffies count

SYNOPSIS

jiffies:long()
ARGUMENTS
None

DESCRIPTION
This function returns the value of the kernel jiffies variable. This value is incremented periodically by timer interrupts, and may wrap around a 32-bit or 64-bit boundary. See `HZ`.

NAME
function::local_clock_ms — Number of milliseconds on the local cpu's clock

SYNOPSIS

```
local_clock_ms:long()
```

ARGUMENTS
None

DESCRIPTION
This function returns the number of milliseconds on the local cpu's clock. This is always monotonic comparing on the same cpu, but may have some drift between cpus (within about a jiffy).

NAME
function::local_clock_ns — Number of nanoseconds on the local cpu's clock

SYNOPSIS

```
local_clock_ns:long()
```

ARGUMENTS
None

DESCRIPTION
This function returns the number of nanoseconds on the local cpu's clock. This is always monotonic comparing on the same cpu, but may have some drift between cpus (within about a jiffy).

NAME
function::local_clock_s — Number of seconds on the local cpu's clock

SYNOPSIS

```
local_clock_s:long()
```

ARGUMENTS
None
DESCRIPTION
This function returns the number of seconds on the local cpu's clock. This is always monotonic comparing on the same cpu, but may have some drift between cpus (within about a jiffy).

NAME
function::local_clock_us — Number of microseconds on the local cpu's clock

SYNOPSIS
    local_clock_us:long()

ARGUMENTS
None

DESCRIPTION
This function returns the number of microseconds on the local cpu's clock. This is always monotonic comparing on the same cpu, but may have some drift between cpus (within about a jiffy).

NAME
function::read_stopwatch_ms — Reads the time in milliseconds for a stopwatch

SYNOPSIS
    read_stopwatch_ms:long(name:string)

ARGUMENTS
name
    stopwatch name

DESCRIPTION
Returns time in milliseconds for stopwatch name. Creates stopwatch name if it does not currently exist.

NAME
function::read_stopwatch_ns — Reads the time in nanoseconds for a stopwatch

SYNOPSIS
    read_stopwatch_ns:long(name:string)

ARGUMENTS
name
stopwatch name

DESCRIPTION
Returns time in nanoseconds for stopwatch name. Creates stopwatch name if it does not currently exist.

NAME
function::read_stopwatch_s — Reads the time in seconds for a stopwatch

SYNOPSIS

read_stopwatch_s:long(name:string)

ARGUMENTS

name
stopwatch name

DESCRIPTION
Returns time in seconds for stopwatch name. Creates stopwatch name if it does not currently exist.

NAME
function::read_stopwatch_us — Reads the time in microseconds for a stopwatch

SYNOPSIS

read_stopwatch_us:long(name:string)

ARGUMENTS

name
stopwatch name

DESCRIPTION
Returns time in microseconds for stopwatch name. Creates stopwatch name if it does not currently exist.

NAME
function::start_stopwatch — Start a stopwatch

SYNOPSIS

start_stopwatch(name:string)
ARGUMENTS

name

the stopwatch name

DESCRIPTION
Start stopwatch name. Creates stopwatch name if it does not currently exist.

NAME

function::stop_stopwatch — Stop a stopwatch

SYNOPSIS

stop_stopwatch(name:string)

ARGUMENTS

name

the stopwatch name

DESCRIPTION
Stop stopwatch name. Creates stopwatch name if it does not currently exist.
CHAPTER 5. TIME UTILITY FUNCTIONS

Utility functions to turn seconds since the epoch (as returned by the timestamp function gettimeofday_s()) into a human readable date/time strings.

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

SYNOPSIS

ctime:string(epochsecs:long)

ARGUMENTS

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

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

"Wed Jun 30 21:49:08 1993"

The string will always be exactly 24 characters. If the time would be unreasonable far in the past (before what can be represented with a 32 bit offset in seconds from the epoch) an error will occur (which can be avoided with try/catch). If the time would be unreasonable far in the future, an error will also occur.

Note that the epoch (zero) corresponds to

"Thu Jan 1 00:00:00 1970"

The earliest full date given by ctime, corresponding to epochsecs -2147483648 is “Fri Dec 13 20:45:52 1901”. The latest full date given by ctime, corresponding to epochsecs 2147483647 is “Tue Jan 19 03:14:07 2038”.


Note that the real C library ctime function puts a newline (\n) character at the end of the string that this function does not. Also note that since the kernel has no concept of timezones, the returned time is always in GMT.

NAME
function::tz_ctime — Convert seconds since epoch into human readable date/time string, with local time zone

SYNOPSIS
tz_ctime(epochsecs:)

ARGUMENTS

epochsecs
    number of seconds since epoch (as returned by gettimeofday_s)

DESCRIPTION
Takes an argument of seconds since the epoch as returned by gettimeofday_s. Returns a string of the same form as ctime, but offsets the epoch time for the local time zone, and appends the name of the local time zone. The string length may vary. The time zone information is passed by staprun at script startup only.

NAME
function::tz_gmtoff — Return local time zone offset

SYNOPSIS

| tz_gmtoff() |

ARGUMENTS
None

DESCRIPTION
Returns the local time zone offset (seconds west of UTC), as passed by staprun at script startup only.

NAME
function::tz_name — Return local time zone name

SYNOPSIS

| tz_name() |

ARGUMENTS
None

DESCRIPTION
Returns the local time zone name, as passed by staprun at script startup only.
CHAPTER 6. SHELL COMMAND FUNCTIONS

Utility functions to enqueue shell commands.

NAME
function::system — Issue a command to the system

SYNOPSIS

system(cmd:string)

ARGUMENTS

cmd

the command to issue to the system

DESCRIPTION
This function runs a command on the system. The command is started in the background some time after the current probe completes. The command is run with the same UID as the user running the stap or staprun command.
CHAPTER 7. MEMORY TAPSET

This family of probe points is used to probe memory-related events or query the memory usage of the current process. It contains the following probe points:

NAME
function::addr_to_node — Returns which node a given address belongs to within a NUMA system

SYNOPSIS

arg_to_node:long(addr:long)

ARGUMENTS

addr
  the address of the faulting memory access

DESCRIPTION
This function accepts an address, and returns the node that the given address belongs to in a NUMA system.

NAME
function::bytes_to_string — Human readable string for given bytes

SYNOPSIS

bytes_to_string:string(bytes:long)

ARGUMENTS

bytes
  Number of bytes to translate.

DESCRIPTION
Returns a string representing the number of bytes (up to 1024 bytes), the number of kilobytes (when less than 1024K) postfixed by ‘K’, the number of megabytes (when less than 1024M) postfixed by ‘M’ or the number of gigabytes postfixed by ‘G’. If representing K, M or G, and the number is amount is less than 100, it includes a ‘.’ plus the remainder. The returned string will be 5 characters wide (padding with whitespace at the front) unless negative or representing more than 9999G bytes.

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

SYNOPSIS
**NAME**
function::pages_to_string — Turns pages into a human readable string

**SYNOPSIS**
```plaintext
pages_to_string: string(pages: long)
```

**ARGUMENTS**
- `pages`
  Number of pages to translate.

**DESCRIPTION**
Multiplies pages by `page_size` to get the number of bytes and returns the result of `bytes_to_string`.

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

**SYNOPSIS**
```plaintext
proc_mem_data: long()
```

**ARGUMENTS**
None

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

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

**SYNOPSIS**
```plaintext
proc_mem_data_pid: long(pid: long)
```

**ARGUMENTS**
- `mem_page_size: long()`
- `pages_to_string: string(pages: long)`
- `proc_mem_data: long()`
- `proc_mem_data_pid: long(pid: long)`
**pid**
The pid of process to examine

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

**NAME**
function::proc_mem_rss — Program resident set size in pages

**SYNOPSIS**

```c
proc_mem_rss:long()
```

**ARGUMENTS**
None

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

**NAME**
function::proc_mem_rss_pid — Program resident set size in pages

**SYNOPSIS**

```c
proc_mem_rss_pid:long(pid:long)
```

**ARGUMENTS**

```
pid
```
The pid of process to examine

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

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

**SYNOPSIS**
ARGUMENTS
None

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

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

SYNOPSIS
proc_mem_shr_pid:long(pid:long)

ARGUMENTS
pid
    The pid of process to examine

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

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

SYNOPSIS
proc_mem_size:long()

ARGUMENTS
None

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

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

SYNOPSIS
ARGUMENTS

pid
The pid of process to examine

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

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

SYNOPSIS

ARGUMENTS
None

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

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

SYNOPSIS

ARGUMENTS

pid
The pid of process to examine

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

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

SYNOPSIS

   proc_mem_txt:long()

ARGUMENTS

None

DESCRIPTION

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

NAME

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

SYNOPSIS

   proc_mem_txt_pid:long(pid:long)

ARGUMENTS

   pid
      The pid of process to examine

DESCRIPTION

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

NAME

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

SYNOPSIS

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

ARGUMENTS

   value
      the fault_type returned by vm.page_fault.return

   test
      the type of fault to test for (VM_FAULT_OOM or similar)
NAME
probe::vm.brk — Fires when a brk is requested (i.e. the heap will be resized)

SYNOPSIS
vm.brk

VALUES
name
name of the probe point

address
the requested address

length
the length of the memory segment

CONTEXT
The process calling brk.

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

SYNOPSIS
vm.kfree

VALUES
name
name of the probe point

ptr
pointer to the kmemory allocated which is returned by kmalloc

caller_function
name of the caller function.

call_site
address of the function calling this kmemory function
NAME
probe::vm.kmalloc — Fires when kmalloc is requested

SYNOPSIS

vm.kmalloc

VALUES

gfp_flags
    type of kmemory to allocate

bytes_req
    requested Bytes

name
    name of the probe point

ptr
    pointer to the kmemory allocated

bytes_alloc
    allocated Bytes

caller_function
    name of the caller function

gfp_flag_name
    type of kmemory to allocate (in String format)

call_site
    address of the kmemory function

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

SYNOPSIS

vm.kmalloc_node

VALUES

caller_function
    name of the caller function
**NAME**
probe::vm.kmem_cache_alloc — Fires when kmem_cache_alloc is requested

**SYNOPSIS**
```
vm.kmem_cache_alloc
```

**VALUES**

`bytes_alloc`
allocated Bytes

`ptr`
pointer to the kmemory allocated

`name`
nname of the probe point

`bytes_req`
requested Bytes

`gfp_flags`
type of kmemory to allocate
**NAME**
probe::vm.kmem_cache_alloc_node — Fires when kmem_cache_alloc_node is requested

**SYNOPSIS**
```
vm.kmem_cache_alloc_node
```

**VALUES**

- **gfp_flags**
  type of kmemory to allocate

- **name**
  name of the probe point

- **bytes_req**
  requested Bytes

- **ptr**
  pointer to the kmemory allocated

- **bytes_alloc**
  allocated Bytes

- **caller_function**
  name of the caller function

- **call_site**
  address of the function calling this kmemory function

- **gfp_flag_name**
  type of kmemory to allocate (in string format)
NAME
probe::vm.kmem_cache_free — Fires when kmem_cache_free is requested

SYNOPSIS

vm.kmem_cache_free

VALUES

caller_function
   Name of the caller function.

call_site
   Address of the function calling this kmemory function

ptr
   Pointer to the kmemory allocated which is returned by kmem_cache

name
   Name of the probe point

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

SYNOPSIS

vm.mmap

VALUES

name
   name of the probe point

length
   the length of the memory segment

address
   the requested address

CONTEXT
   The process calling mmap.
NAME
probe::vm.munmap — Fires when an munmap is requested

SYNOPSIS
vm.munmap

VALUES
length
  the length of the memory segment

address
  the requested address

name
  name of the probe point

CONTEXT
The process calling munmap.

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

SYNOPSIS
vm.oom_kill

VALUES
name
  name of the probe point

task
  the task being killed

CONTEXT
The process that tried to consume excessive memory, and thus triggered the OOM.

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

SYNOPSIS
vm.pagefault

VALUES

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

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

name
   name of the probe point

CONTEXT
The process which triggered the fault

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

SYNOPSIS

vm.pagefault.return

VALUES

name
   name of the probe point

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

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

SYNOPSIS

vm.write_shared

VALUES

address
the address of the shared write

\textit{name}

name of the probe point

\textbf{CONTEXT}
The context is the process attempting the write.

\textbf{DESCRIPTION}
Fires when a process attempts to write to a shared page. If a copy is necessary, this will be followed by a \texttt{vm.write_shared_copy}.

\textbf{NAME}
probe::vm.write_shared_copy — Page copy for shared page write

\textbf{SYNOPSIS}

\begin{verbatim}
vm.write_shared_copy
\end{verbatim}

\textbf{VALUES}

\begin{description}
  \item [zero] boolean indicating whether it is a zero page (can do a clear instead of a copy)
  \item [name] Name of the probe point
  \item [address] The address of the shared write
\end{description}

\textbf{CONTEXT}
The process attempting the write.

\textbf{DESCRIPTION}
Fires when a write to a shared page requires a page copy. This is always preceded by a \texttt{vm.write_shared}.
CHAPTER 8. TASK TIME TAPSET

This tapset defines utility functions to query time related properties of the current tasks, translate those in milliseconds and human readable strings.

NAME

function::cputime_to_msecs — Translates the given cputime into milliseconds

SYNOPSIS

| cputime_to_msecs:long(cputime:long) |

ARGUMENTS

\( cputime \)

Time to convert to milliseconds.

NAME

function::cputime_to_string — Human readable string for given cputime

SYNOPSIS

| cputime_to_string:string(cputime:long) |

ARGUMENTS

\( cputime \)

Time to translate.

DESCRIPTION

Equivalent to calling: msec_to_string (cputime_to_msecs (cputime)).

NAME

function::cputime_to_usecs — Translates the given cputime into microseconds

SYNOPSIS

| cputime_to_usecs:long(cputime:long) |

ARGUMENTS

\( cputime \)

Time to convert to microseconds.
NAME
function::msecs_to_string — Human readable string for given milliseconds

SYNOPSIS

msecs_to_string:string(msecs:long)

ARGUMENTS

msecs
    Number of milliseconds to translate.

DESCRIPTION
Returns a string representing the number of milliseconds as a human readable string consisting of 
“XmY.ZZZs”, where X is the number of minutes, Y is the number of seconds and ZZZ is the number of 
milliseconds.

NAME
function::nsecs_to_string — Human readable string for given nanoseconds

SYNOPSIS

nsecs_to_string:string(nsecs:long)

ARGUMENTS

nsecs
    Number of nanoseconds to translate.

DESCRIPTION
Returns a string representing the number of nanoseconds as a human readable string consisting of 
“XmY.ZZZZZZs”, where X is the number of minutes, Y is the number of seconds and ZZZZZZZZZ is the 
number of nanoseconds.

NAME
function::task_start_time — Start time of the given task

SYNOPSIS

task_start_time:long(tid:long)

ARGUMENTS
**tid**

Thread id of the given task

**DESCRIPTION**

Returns the start time of the given task in nanoseconds since boot time or 0 if the task does not exist.

**NAME**

function::task_stime — System time of the current task

**SYNOPSIS**

| task_stime:long() |

**ARGUMENTS**

None

**DESCRIPTION**

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

**NAME**

function::task_stime_tid — System time of the given task

**SYNOPSIS**

| task_stime_tid:long(tid:long) |

**ARGUMENTS**

**tid**

Thread id of the given task

**DESCRIPTION**

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

**NAME**

function::task_time_string — Human readable string of task time usage

**SYNOPSIS**

| task_time_string:string() |
ARGUMENTS
None

DESCRIPTION
Returns a human readable string showing the user and system time the current task has used up to now. For example “usr: 0m12.908s, sys: 1m6.851s”.

NAME
class::task_time_string_tid — Human readable string of task time usage

SYNOPSIS

ARGUMENTS
tid
Thread id of the given task

DESCRIPTION
Returns a human readable string showing the user and system time the given task has used up to now. For example “usr: 0m12.908s, sys: 1m6.851s”.

NAME
class::task_utime — User time of the current task

SYNOPSIS

ARGUMENTS
None

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

NAME
class::task_utime_tid — User time of the given task

SYNOPSIS

ARGUMENTS
tid
Thread id of the given task
ARGUMENTS

\textit{tid}
Thread id of the given task

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

NAME
\texttt{function::usecs\_to\_string} — Human readable string for given microseconds

SYNOPSIS

\begin{verbatim}
usecs\_to\_string:string(usecs:long)
\end{verbatim}

ARGUMENTS

\textit{usecs}
Number of microseconds to translate.

DESCRIPTION
Returns a string representing the number of microseconds as a human readable string consisting of “XmY.ZZZZZZs”, where X is the number of minutes, Y is the number of seconds and ZZZZZZ is the number of microseconds.
CHAPTER 9. SCHEDULER TAPSET

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

NAME
probe::scheduler.balance — A cpu attempting to find more work.

SYNOPSIS

\[
\text{scheduler.balance}
\]

VALUES

name
name of the probe point

CONTEXT
The cpu looking for more work.

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

SYNOPSIS

\[
\text{scheduler.cpu_off}
\]

VALUES

\begin{itemize}
  \item \texttt{task\_prev}
    \begin{itemize}
      \item the process leaving the cpu (same as current)
    \end{itemize}
  \item \texttt{idle}
    \begin{itemize}
      \item boolean indicating whether current is the idle process
    \end{itemize}
  \item \texttt{name}
    \begin{itemize}
      \item name of the probe point
    \end{itemize}
  \item \texttt{task\_next}
    \begin{itemize}
      \item the process replacing current
    \end{itemize}
\end{itemize}

CONTEXT
The process leaving the cpu.
NAME
probe::scheduler.cpu_on — Process is beginning execution on a cpu

SYNOPSIS

scheduler.cpu_on

VALUES

idle
- boolean indicating whether current is the idle process

task_prev
the process that was previously running on this cpu

name
name of the probe point

CONTEXT
The resuming process.

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

SYNOPSIS

scheduler.ctxswitch

VALUES

prev_tid
The TID of the process to be switched out

name
name of the probe point

next_tid
The TID of the process to be switched in

prev_pid
The PID of the process to be switched out

prevtsk_state
the state of the process to be switched out
**next_pid**
The PID of the process to be switched in

**nexttsk_state**
the state of the process to be switched in

**prev_priority**
The priority of the process to be switched out

**next_priority**
The priority of the process to be switched in

**prev_task_name**
The name of the process to be switched out

**next_task_name**
The name of the process to be switched in

---

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

**SYNOPSIS**
scheduler.kthread_stop

**VALUES**

**thread_pid**
PID of the thread being stopped

**thread_priority**
priority of the thread

---

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

**SYNOPSIS**
scheduler.kthread_stop.return

**VALUES**

**return_value**
return value after stopping the thread

name
  name of the probe point

NAME
probe::scheduler.migrate — Task migrating across cpus

SYNOPSIS

  scheduler.migrate

VALUES

priority
  priority of the task being migrated

cpu_to
  the destination cpu

cpu_from
  the original cpu

task
  the process that is being migrated

name
  name of the probe point

pid
  PID of the task being migrated

NAME
probe::scheduler.process_exit — Process exiting

SYNOPSIS

  scheduler.process_exit

VALUES

name
NAME
probe::scheduler.process_fork — Process forked

SYNOPSIS

| scheduler.process_fork |

VALUES

name
name of the probe point

parent_pid
PID of the parent process

child_pid
PID of the child process

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

SYNOPSIS

| scheduler.process_free |

VALUES

name
name of the probe point

pid
PID of the process getting freed

priority
priority of the process getting freed

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

SYNOPSIS

```
scheduler.process_wait
```

VALUES

- **name**
  - name of the probe point

- **pid**
  - PID of the process scheduler is waiting on

NAME
probe::scheduler.signal_send — Sending a signal

SYNOPSIS

```
scheduler.signal_send
```

VALUES

- **pid**
  - pid of the process sending signal

- **name**
  - name of the probe point

- **signal_number**
  - signal number

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

SYNOPSIS
scheduler.tick

VALUES

idle
boolean indicating whether current is the idle process

name
name of the probe point

CONTEXT
The process whose accounting will be updated.

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

SYNOPSIS

scheduler.wait_task

VALUES

task_pid
PID of the task the scheduler is waiting on

name
name of the probe point

task_priority
priority of the task

NAME
probe::scheduler.wakeup — Task is woken up

SYNOPSIS

scheduler.wakeup

VALUES

task_tid
tid of the task being woken up
**NAME**

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

**SYNOPSIS**

```plaintext
scheduler.wakeup_new
```

**VALUES**

- **name**: name of the probe point
- **task_state**: state of the task woken up
- **task_pid**: PID of the new task woken up
- **task_tid**: TID of the new task woken up
- **task_priority**: priority of the new task
- **task_cpu**: cpu of the task woken up
CHAPTER 10. IO SCHEDULER AND BLOCK IO TAPSET

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

**NAME**

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

**SYNOPSIS**

ioblock.end

**VALUES**

- **name**
  - name of the probe point

- **sector**
  - beginning sector for the entire bio

- **hw_segments**
  - number of segments after physical and DMA remapping hardware coalescing is performed

- **phys_segments**
  - number of segments in this bio after physical address coalescing is performed.

- **flags**
  - see below BIO_UPTODATE 0 ok after I/O completion BIO_RW_BLOCK 1 RW_AHEAD set, and read/write would block BIO_EOF 2 out-out-bounds error BIO_SEG_VALID 3 nr_hw_seg valid BIO_CLONED 4 doesn’t own data BIO_BOUNCED 5 bio is a bounce bio BIO_USER_MAPPED 6 contains user pages BIO_EOPNOTSUPP 7 not supported

- **devname**
  - block device name

- **bytes_done**
  - number of bytes transferred

- **error**
  - 0 on success

- **size**
  - total size in bytes

- **idx**
  - offset into the bio vector array
vcnt
bio vector count which represents number of array element (page, offset, length) which makes up this I/O request

ino
i-node number of the mapped file

rw
binary trace for read/write request

CONTEXT
The process signals the transfer is done.

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

SYNOPSIS

VALUES
sector
beginning sector for the entire bio

name
name of the probe point

devname
block device name

phys_segments
number of segments in this bio after physical address coalescing is performed

flags
see below BIO_UPTODATE 0 ok after I/O completion BIO_RW_BLOCK 1 RW_AHEAD set, and read/write would block BIO_EOF 2 out-out-bounds error BIO_SEG_VALID 3 nr_hw_seg valid BIO_CLONED 4 doesn't own data BIO_BOUNCED 5 bio is a bounce bio BIO_USER_MAPPED 6 contains user pages BIO_EOPNOTSUPP 7 not supported

hw_segments
number of segments after physical and DMA remapping hardware coalescing is performed

bdev_contains
points to the device object which contains the partition (when bio structure represents a partition)
vcnt
  bio vector count which represents number of array element (page, offset, length) which make up this I/O request

idx
  offset into the bio vector array

bdev
  target block device

p_start_sect
  points to the start sector of the partition structure of the device

size
  total size in bytes

ino
  i-node number of the mapped file

rw
  binary trace for read/write request

CONTEXT
The process makes block I/O request

NAME
probe::ioblock_trace.bounce — Fires whenever a buffer bounce is needed for at least one page of a block I/O request.

SYNOPSIS

ioblock_trace.bounce

VALUES

q
  request queue on which this bio was queued.

size
  total size in bytes

vcnt
  bio vector count which represents number of array element (page, offset, length) which makes up this I/O request
idx
offset into the bio vector array phys_segments - number of segments in this bio after physical address coalescing is performed.

bdev
target block device

p_start_sect
points to the start sector of the partition structure of the device

ino
i-node number of the mapped file

rw
binary trace for read/write request

name
name of the probe point

sector
beginning sector for the entire bio

bdev_contains
points to the device object which contains the partition (when bio structure represents a partition)

devname
device for which a buffer bounce was needed.

flags
see below BIO_UPTODATE 0 ok after I/O completion BIO_RW_BLOCK 1 RW_AHEAD set, and read/write would block BIO_EOF 2 out-out-bounds error BIO_SEG_VALID 3 nr_hw_seg valid BIO_CLONED 4 doesn't own data BIO_BOUNCED 5 bio is a bounce bio BIO_USER_MAPPED 6 contains user pages BIO_EOPNOTSUPP 7 not supported

bytes_done
number of bytes transferred

CONTEXT
The process creating a block IO request.

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

SYNOPSIS
-
VALUES

*bdev_contains*

points to the device object which contains the partition (when bio structure represents a partition)

*flags*

see below BIO_UPTODATE 0 ok after I/O completion BIO_RW_BLOCK 1 RW_AHEAD set, and read/write would block BIO_EOF 2 out-out-bounds error BIO_SEG_VALID 3 nr_hw_seg valid BIO_CLONED 4 doesn't own data BIO_BOUNCED 5 bio is a bounce bio BIO_USER_MAPPED 6 contains user pages BIO_EOPNOTSUPP 7 not supported

*devname*

block device name

*bytes_done*

number of bytes transferred

*name*

name of the probe point

*sector*

beginning sector for the entire bio

*ino*

i-node number of the mapped file

*rw*

binary trace for read/write request

*size*

total size in bytes

*q*

request queue on which this bio was queued.

*idx*

offset into the bio vector array *phys_segments* - number of segments in this bio after physical address coalescing is performed.

*vcnt*

bio vector count which represents number of array element (page, offset, length) which makes up this I/O request

*bdev*
target block device

\textbf{p\_start\_sect}
points to the start sector of the partition structure of the device

\textbf{CONTEXT}
The process signals the transfer is done.

\textbf{NAME}
probe::ioblock\_trace.request — Fires just as a generic block I/O request is created for a bio.

\textbf{SYNOPSIS}

\begin{verbatim}
ioblock_trace.request
\end{verbatim}

\textbf{VALUES}

\begin{itemize}
\item \textit{q}
  request queue on which this bio was queued.
\item \textit{size}
  total size in bytes
\item \textit{idx}
  offset into the bio vector array \texttt{phys\_segments} - number of segments in this bio after physical address coalescing is performed.
\item \textit{vcnt}
  bio vector count which represents number of array element (page, offset, length) which make up this I/O request
\item \textit{bdev}
  target block device
\item \textit{p\_start\_sect}
  points to the start sector of the partition structure of the device
\item \textit{ino}
  i-node number of the mapped file
\item \textit{rw}
  binary trace for read/write request
\item \textit{name}
  name of the probe point
\end{itemize}
sector
beginning sector for the entire bio

bdev_contains
points to the device object which contains the partition (when bio structure represents a partition)

devname
block device name

flags
see below BIO_UPTODATE 0 ok after I/O completion BIO_RW_BLOCK 1 RW_AHEAD set, and read/write would block BIO_EOF 2 out-out-bounds error BIO_SEG_VALID 3 nr_hw_seg valid BIO_CLONED 4 doesn't own data BIO_BOUNCED 5 bio is a bounce bio BIO_USER_MAPPED 6 contains user pages BIO_EOPNOTSUPP 7 not supported

bytes_done
number of bytes transferred

CONTEXT
The process makes block I/O request

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

SYNOPSIS

ioscheduler.elv_add_request

VALUES

rq
Address of request.

q
Pointer to request queue.

elevator_name
The type of I/O elevator currently enabled.

disk_major
Disk major no of request.

disk_minor
Disk minor number of request.
rq_flags
Request flags.

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

SYNOPSIS
ioscheduler.elv_add_request.kp

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

NAME
probe::ioscheduler.elv_add_request.tp — tracepoint based probe to indicate a request is added to the request queue.

SYNOPSIS
ioscheduler.elv_add_request.tp
VALUES

q
Pointer to request queue.

elevator_name
The type of I/O elevator currently enabled.

name
Name of the probe point

rq
Address of request.

disk_major
Disk major no of request.

disk_minor
Disk minor number of request.

rq_flags
Request flags.

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

SYNOPSIS

ioscheduler.elv_completed_request

VALUES

name
Name of the probe point

rq
Address of the request

elevator_name
The type of I/O elevator currently enabled

disk_major
Disk major number of the request
**NAME**
probe::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
   Name of the probe point

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

**SYNOPSIS**
ioscheduler.elv_next_request.return

**VALUES**

disk_major
   Disk major number of the request

disk_minor
   Disk minor number of the request

rq_flags
   Request flags

rq
   Address of the request
**NAME**
probe::ioscheduler_trace.elv_abort_request — Fires when a request is aborted.

**SYNOPSIS**
- `ioscheduler_trace.elv_abort_request`

**VALUES**
- `disk_major`
  Disk major no of request.
- `disk_minor`
  Disk minor number of request.
- `rq_flags`
  Request flags.
- `elevator_name`
  The type of I/O elevator currently enabled.
- `rq`
  Address of request.

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

**SYNOPSIS**
- `ioscheduler_trace.elv_completed_request`

**VALUES**
- `elevator_name`
  The type of I/O elevator currently enabled.
### NAME
probe::ioscheduler_trace.elv_issue_request — Fires when a request is

### SYNOPSIS

```c
ioscheduler_trace.elv_issue_request
```

### VALUES

- **rq Flags**
  - Request flags.

- **disk_minor**
  - Disk minor number of request.

- **disk_major**
  - Disk major no of request.

- **elevator_name**
  - The type of I/O elevator currently enabled.

- **rq**
  - Address of request.

- **name**
  - Name of the probe point
DESCRIPTION
scheduled.

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

SYNOPSIS
ioscheduler_trace.elv_requeue_request

VALUES

rq
Address of request.

name
Name of the probe point

elevator_name
The type of I/O elevator currently enabled.

rq_flags
Request flags.

disk_minor
Disk minor number of request.

disk_major
Disk major no of request.

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

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

SYNOPSIS
ioscheduler_trace.plug

VALUES

rq_queue
request queue

name
Name of the probe point

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

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

SYNOPSIS
ioscheduler_trace.unplug_io

VALUES
name
Name of the probe point

rq_queue
request queue

DESCRIPTION
Either, when number of pending requests in the queue exceeds threshold or, upon expiration of timer that was activated when queue was plugged.

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

SYNOPSIS
ioscheduler_trace.unplug_timer

VALUES
rq_queue
request queue

name
Name of the probe point

DESCRIPTION
with a request queue expires.
CHAPTER 11. SCSI TAPSET

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

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

SYNOPSIS

sas.iocompleted

VALUES

**device_state**
The current state of the device

**dev_id**
The scsi device id

**req_addr**
The current struct request pointer, as a number

**data_direction_str**
Data direction, as a string

**device_state_str**
The current state of the device, as a string

**lun**
The lun number

**goodbytes**
The bytes completed

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

**channel**
The channel number

**host_no**
The host number
NAME
probe::scsi.iodispatching — SCSI mid-layer dispatched low-level SCSI command

SYNOPSIS

scsi.iodispatching

VALUES

device_state
    The current state of the device

request_bufflen
    The request buffer length

request_buffer
    The request buffer address

dev_id
    The scsi device id

data_direction_str
    Data direction, as a string

req_addr
    The current struct request pointer, as a number

device_state_str
    The current state of the device, as a string

lun
    The lun 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)

channel
    The channel number

host_no
    The host number
probe::scsi.iiodone — SCSI command completed by low level driver and enqueued into the done queue.

**SYNOPSIS**

```c
scsi.iiodone
```

**VALUES**

- `device_state`
  - The current state of the device

- `data_direction_str`
  - Data direction, as a string

- `req_addr`
  - The current struct request pointer, as a number

- `dev_id`
  - The scsi device id

- `lun`
  - The lun number

- `scsi_timer_pending`
  - 1 if a timer is pending on this request

- `device_state_str`
  - The current state of the device, as a string

- `host_no`
  - The host number

- `channel`
  - The channel number

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

**NAME**

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

**SYNOPSIS**

```c
scsi.ioentry
```
VALUES

req_addr
The current struct request pointer, as a number

disk_major
The major number of the disk (-1 if no information)

device_state_str
The current state of the device, as a string

disk_minor
The minor number of the disk (-1 if no information)

device_state
The current state of the device

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

SYNOPSIS

scsi.ioexecute

VALUES

host_no
The host number

channel
The channel number

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

lun
The lun number

retries
Number of times to retry request

device_state_str
The current state of the device, as a string
**NAME**
probe::scsi.set_state — Order SCSI device state change

**SYNOPSIS**
```
scsi.set_state
```

**VALUES**

**state**
The new state of the device

**old_state**
The current state of the device

**dev_id**
The scsi device id

**state_str**
The new state of the device, as a string

**old_state_str**
The current state of the device, as a string

**lun**
The lun number
**channel**

The channel number

**host_no**

The host number
CHAPTER 12. TTY TAPSET

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

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

SYNOPSIS

```
tty.init
```

VALUES

- **name**
  the driver .dev_name name

- **module**
  the module name

- **driver_name**
  the driver name


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

SYNOPSIS

```
tty.ioctl
```

VALUES

- **arg**
  the ioctl argument

- **name**
  the file name

- **cmd**
  the ioctl command
probe::tty.open — Called when a tty is opened

SYNOPSIS

| tty.open |

VALUES

inode_state
  the inode state

file_mode
  the file mode

inode_number
  the inode number

file_flags
  the file flags

file_name
  the file name

inode_flags
  the inode flags

NAME

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

SYNOPSIS

| tty.poll |

VALUES

file_name
  the tty file name

wait_key
  the wait queue key

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

SYNOPSIS

tty.read

VALUES

file_name
the file name treated to the tty

driver_name
the driver name

nr
The amount of characters to be read

buffer
the buffer that will receive the characters

NAME

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

SYNOPSIS

tty.receive

VALUES

driver_name
the driver name

count
The amount of characters received

index
The tty Index

cp
the buffer that was received

id
the tty id

name
the name of the module file

fp

The flag buffer

---

**NAME**

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

**SYNOPSIS**

```
tty.register
```

**VALUES**

- `name`
  - the driver .dev_name name

- `module`
  - the module name

- `index`
  - the tty index requested

- `driver_name`
  - the driver name

---

**NAME**

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

**SYNOPSIS**

```
tty.release
```

**VALUES**

- `inode_flags`
  - the inode flags

- `file_flags`
  - the file flags

- `file_name`
the file name

inode_state
  the inode state

inode_number
  the inode number

file_mode
  the file mode

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

SYNOPSIS
  tty.resize

VALUES
new_row
  the new row value

old_row
  the old row value

name
  the tty name

new_col
  the new col value

old_xpixel
  the old xpixel

old_col
  the old col value

new_xpixel
  the new xpixel value

old_ypixel
  the old ypixel
**NAME**
probe::tty.unregister — Called when a tty device is being unregistered

**SYNOPSIS**

```
tty.unregister
```

**VALUES**

- **name**
  the driver .dev_name name

- **module**
  the module name

- **index**
  the tty index requested

- **driver_name**
  the driver name

---

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

**SYNOPSIS**

```
tty.write
```

**VALUES**

- **nr**
  The amount of characters

- **buffer**
  the buffer that will be written

- **file_name**
  the file name treated to the tty
driver_name

the driver name
CHAPTER 13. INTERRUPT REQUEST (IRQ) TAPSET

This family of probe points is used to probe interrupt request (IRQ) activities. It contains the following probe points:

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

SYNOPSIS

| irq_handler.entry |

VALUES

next_irqaction
pointer to next irqaction for shared interrupts

dirty_fn
interrupt handler function for threaded interrupts

dirty
thread pointer for threaded interrupts

dirty_flags
Flags related to thread

irq
irq number
lags_str
symbolic string representation of IRQ flags

dev_name
name of device

action
struct irqaction* for this interrupt num

dir
pointer to the proc/irq/NN/name entry

flags
Flags for IRQ handler

dev_id
Cookie to identify device
**handler**

interrupt handler function

---

**NAME**

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

**SYNOPSIS**

```
irq_handler.exit
```

**VALUES**

- **flags_str**
  symbolic string representation of IRQ flags

- **dev_name**
  name of device

- **ret**
  return value of the handler

- **action**
  `struct irqaction*`

- **thread_fn**
  interrupt handler function for threaded interrupts

- **next_irqaction**
  pointer to next irqaction for shared interrupts

- **thread**
  thread pointer for threaded interrupts

- **thread_flags**
  Flags related to thread

- **irq**
  interrupt number

- **handler**
  interrupt handler function that was executed

- **flags**
  flags for IRQ handler
**NAME**
probe::softirq.entry — Execution of handler for a pending softirq starting

**SYNOPSIS**
```
softirq.entry
```

**VALUES**

- **action**
  pointer to softirq handler just about to execute

- **vec_nr**
  softirq vector number

- **vec**
  softirq_action vector

- **h**
  struct softirq_action* for current pending softirq

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

**SYNOPSIS**
```
softirq.exit
```

**VALUES**

- **vec_nr**
  softirq vector number

- **action**
  pointer to softirq handler that just finished execution
NAME
probe::workqueue.create — Creating a new workqueue

SYNOPSIS

workqueue.create

VALUES

wq_thread
task_struct of the workqueue thread

cpu
cpu for which the worker thread is created

NAME
probe::workqueue.destroy — Destroying workqueue

SYNOPSIS

workqueue.destroy

VALUES

wq_thread
task_struct of the workqueue thread

NAME
probe::workqueue.execute — Executing deferred work

SYNOPSIS

workqueue.execute
VALUES

\textit{wq\_thread}

\textit{task\_struct} of the workqueue thread

\textit{work\_func}

pointer to handler function

\textit{work}

\textit{work\_struct\*} being executed

---

NAME

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

SYNOPSIS

\begin{verbatim}
workqueue.insert
\end{verbatim}

VALUES

\textit{wq\_thread}

\textit{task\_struct} of the workqueue thread

\textit{work\_func}

pointer to handler function

\textit{work}

\textit{work\_struct\*} being queued
CHAPTER 14. NETWORKING TAPSET

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

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

SYNOPSIS

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

ARGUMENTS

addr
  the IP address

family
  the IP address family (either AF_INET or AF_INET6)

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

SYNOPSIS

htonl::long(x:long)

ARGUMENTS

x
  Value to convert

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

SYNOPSIS

htonll::long(x:long)

ARGUMENTS

x
  Value to convert
NAME
function::htons — Convert 16-bit short from host to network order

SYNOPSIS

htons:long(x:long)

ARGUMENTS

x
Value to convert

NAME
function::ip_ntop — Returns a string representation for an IPv4 address

SYNOPSIS

ip_ntop:string(addr:long)

ARGUMENTS

addr
the IPv4 address represented as an integer

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

SYNOPSIS

ntohl:long(x:long)

ARGUMENTS

x
Value to convert

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

```
ntholl:long(x:long)
```

ARGUMENTS

```
x
   Value to convert
```

NAME

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

SYNOPSIS

```
nthiol:long(x:long)
```

ARGUMENTS

```
x
   Value to convert
```

NAME

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

SYNOPSIS

```
netdev.change_mac
```

VALUES

```
mac_len
   The MAC length

old_mac
   The current MAC address

dev_name
   The device that will have the MAC changed

new_mac
   The new MAC address
```
NAME
probe::netdev.change_mtu — Called when the netdev MTU is changed

SYNOPSIS

netdev.change_mtu

VALUES

old_mtu
    The current MTU

new_mtu
    The new MTU

dev_name
    The device that will have the MTU changed

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

SYNOPSIS

netdev.change_rx_flag

VALUES

flags
    The new flags

dev_name
    The device that will be changed

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

SYNOPSIS

netdev.close

VALUES

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

SYNOPSIS
netdev.get_stats

VALUES

dev_name
   The device that is going to provide the statistics

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

SYNOPSIS
netdev.hard_transmit

VALUES

truesize
   The size of the data to be transmitted.

dev_name
   The device scheduled to transmit

protocol
   The protocol used in the transmission

length
   The length of the transmit buffer.

NAME
probe::netdev.ioctl — Called when the device suffers an IOCTL

SYNOPSIS
netdev.ioctl

VALUES

arg
The IOCTL argument (usually the netdev interface)

cmd
The IOCTL request

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

SYNOPSIS

netdev.open

VALUES

dev_name
The device that is going to be opened

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

SYNOPSIS

netdev.receive

VALUES

length
The length of the receiving buffer.

protocol
Protocol of received packet.

dev_name
The name of the device. e.g: eth0, ath1.
NAME
probe::netdev.register — Called when the device is registered

SYNOPSIS

netdev.register

VALUES

dev_name
   The device that is going to be registered

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

SYNOPSIS

netdev.rx

VALUES

dev_name
   The device received the packet

protocol
   The packet protocol

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

SYNOPSIS

netdev.set_promiscuity

VALUES

dev_name
   The device that is entering/leaving promiscuity mode

enable
   If the device is entering promiscuity mode

inc
Count the number of promiscuity openers

**disable**
If the device is leaving promiscuity mode

---

**NAME**
probe::netdev.transmit — Network device transmitting buffer

**SYNOPSIS**

```plaintext
netdev.transmit
```

**VALUES**

- **protocol**
  - The protocol of this packet (defined in `include/linux/if_ether.h`).

- **length**
  - The length of the transmit buffer.

- **truesize**
  - The size of the data to be transmitted.

- **dev_name**
  - The name of the device. e.g: eth0, ath1.

---

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

**SYNOPSIS**

```plaintext
netdev.unregister
```

**VALUES**

- **dev_name**
  - The device that is going to be unregistered

---

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

---

182
SYNOPSIS

netfilter.arp.forward

VALUES

ar_hln
Length of hardware address

nf_stop
Constant used to signify a 'stop' verdict

outdev_name
Name of network device packet will be routed to (if known)

ar_tha
Ethernet+IP only (ar_pro==0x800): target hardware (MAC) address

nf_accept
Constant used to signify an 'accept' verdict

ar_data
Address of ARP packet data region (after the header)

indev_name
Name of network device packet was received on (if known)

arphdr
Address of ARP header

outdev
Address of net_device representing output device, 0 if unknown

nf_repeat
Constant used to signify a 'repeat' verdict

length
The length of the packet buffer contents, in bytes

nf_stolen
Constant used to signify a 'stolen' verdict

ar_pln
Length of protocol address

pf
Protocol family -- always "arp"

**ar_sha**
Ethernet+IP only (ar_pro==0x800): source hardware (MAC) address

**indiv**
Address of net_device representing input device, 0 if unknown

**nf_drop**
Constant used to signify a 'drop' verdict

**ar_pro**
Format of protocol address

**ar_sip**
Ethernet+IP only (ar_pro==0x800): source IP address

**ar_tip**
Ethernet+IP only (ar_pro==0x800): target IP address

**ar_hrd**
Format of hardware address

**nf_queue**
Constant used to signify a 'queue' verdict

**ar_op**
ARP opcode (command)

---

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

**SYNOPSIS**

```
netfilter.arp.in
```

**VALUES**

**ar_hln**
Length of hardware address

**nf_stop**
Constant used to signify a 'stop' verdict
nf_accept
Constant used to signify an 'accept' verdict

ar_tha
Ethernet+IP only (ar_pro==0x800): target hardware (MAC) address

ar_data
Address of ARP packet data region (after the header)

outdev_name
Name of network device packet will be routed to (if known)

outdev
Address of net_device representing output device, 0 if unknown

nf_repeat
Constant used to signify a 'repeat' verdict

arphdr
Address of ARP header

indev_name
Name of network device packet was received on (if known)

nf_stolen
Constant used to signify a 'stolen' verdict

length
The length of the packet buffer contents, in bytes

ar_pln
Length of protocol address

ar_sha
Ethernet+IP only (ar_pro==0x800): source hardware (MAC) address

pf
Protocol family -- always “arp”

nf_drop
Constant used to signify a 'drop' verdict

ar_pro
Format of protocol address
**NAME**
probe::netfilter.arp.out — Called for each outgoing ARP packet

**SYNOPSIS**

```
netfilter.arp.out
```

**VALUES**

**ar_tip**
Ethernet+IP only (ar_pro==0x800): target IP address

**nf_drop**
Constant used to signify a 'drop' verdict

**ar_pro**
Format of protocol address

**ar_sip**
Ethernet+IP only (ar_pro==0x800): source IP address

**indev**
Address of net_device representing input device, 0 if unknown

**ar_sha**
Ethernet+IP only (ar_pro==0x800): source hardware (MAC) address
pf
Protocol family -- always “arp”

ar_op
ARP opcode (command)

nf_queue
Constant used to signify a ‘queue’ verdict

ar_hrd
Format of hardware address

nf_accept
Constant used to signify an 'accept' verdict

ar_data
Address of ARP packet data region (after the header)

ar_tha
Ethernet+IP only (ar_pro==0x800): target hardware (MAC) address

outdev_name
Name of network device packet will be routed to (if known)

nf_stop
Constant used to signify a 'stop' verdict

ar_hln
Length of hardware address

ar_pln
Length of protocol address

nf_stolen
Constant used to signify a 'stolen' verdict

length
The length of the packet buffer contents, in bytes

outdev
Address of net_device representing output device, 0 if unknown

nf_repeat
Constant used to signify a 'repeat' verdict
**NAME**
probe::netfilter.bridge.forward — Called on an incoming bridging packet destined for some other computer

**SYNOPSIS**

```
netfilter.bridge.forward
```

**VALUES**

- `br_fd`
  Forward delay in 1/256 secs

- `nf_queue`
  Constant used to signify a 'queue' verdict

- `brhdr`
  Address of bridge header

- `br_mac`
  Bridge MAC address

- `indev`
  Address of net_device representing input device, 0 if unknown

- `br_msg`
  Message age in 1/256 secs

- `nf_drop`
  Constant used to signify a 'drop' verdict

- `llcproto_stp`
  Constant used to signify Bridge Spanning Tree Protocol packet

- `pf`
  Protocol family -- always "bridge"

- `br_vid`
Protocol version identifier

*indev_name*
Name of network device packet was received on (if known)

*br_poid*
Port identifier

*outdev*
Address of net_device representing output device, 0 if unknown

*nf_repeat*
Constant used to signify a 'repeat' verdict

*llcpdu*
Address of LLC Protocol Data Unit

*length*
The length of the packet buffer contents, in bytes

*nf_stolen*
Constant used to signify a 'stolen' verdict

*br_cost*
Total cost from transmitting bridge to root

*nf_stop*
Constant used to signify a 'stop' verdict

*br_type*
BPDU type

*br_max*
Max age in 1/256 secs

*br_htime*
Hello time in 1/256 secs

*protocol*
Packet protocol

*br_bid*
Identity of bridge

*br_rmac*
Root bridge MAC address

**br_prid**
Protocol identifier

**outdev_name**
Name of network device packet will be routed to (if known)

**br_flags**
BPDU flags

**nf_accept**
Constant used to signify an 'accept' verdict

**br_rid**
Identity of root bridge

---

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

**SYNOPSIS**

```text
netfilter.bridge.local_in
```

**VALUES**

**llcproto_stp**
Constant used to signify Bridge Spanning Tree Protocol packet

**pf**
Protocol family -- always “bridge”

**nf_drop**
Constant used to signify a 'drop' verdict

**br_msg**
Message age in 1/256 secs

**indev**
Address of net_device representing input device, 0 if unknown

**nf_queue**
Constant used to signify a 'queue' verdict
**br_fd**
Forward delay in 1/256 secs

**br_mac**
Bridge MAC address

**brhdr**
Address of bridge header

**br_rid**
Identity of root bridge

**nf_accept**
Constant used to signify an 'accept' verdict

**outdev_name**
Name of network device packet will be routed to (if known)

**br_flags**
BPDU flags

**br_prid**
Protocol identifier

**br_hstime**
Hello time in 1/256 secs

**protocol**
Packet protocol

**br_bid**
Identity of bridge

**br_rmac**
Root bridge MAC address

**br_max**
Max age in 1/256 secs

**br_type**
BPDU type

**nf_stop**
Constant used to signify a 'stop' verdict
### NAME

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

### SYNOPSIS

```
netfilter.bridge.local_out
```

### VALUES

- **indev**
  - Address of net_device representing input device, 0 if unknown

- **br_msg**
  - Message age in 1/256 secs

- **nf_drop**
  - Constant used to signify a ‘drop’ verdict
**llcproto_stp**
Constant used to signify Bridge Spanning Tree Protocol packet

**pf**
Protocol family -- always "bridge"

**br_fd**
Forward delay in 1/256 secs

**nf_queue**
Constant used to signify a 'queue' verdict

**brhdr**
Address of bridge header

**br_mac**
Bridge MAC address

**br_flags**
BPDU flags

**outdev_name**
Name of network device packet will be routed to (if known)

**nf_accept**
Constant used to signify an 'accept' verdict

**br_rid**
Identity of root bridge

**nf_stop**
Constant used to signify a 'stop' verdict

**br_type**
BPDU type

**br_max**
Max age in 1/256 secs

**protocol**
Packet protocol

**br_hptime**
Hello time in 1/256 secs
**NAME**
probe::netfilter.bridge.post_routing — Called before a bridging packet hits the wire

**SYNOPSIS**

```plaintext
netfilter.bridge.post_routing
```
VALUES

1lcproto_stp
    Constant used to signify Bridge Spanning Tree Protocol packet

pf
    Protocol family -- always “bridge”

indev
    Address of net_device representing input device, 0 if unknown

nf_drop
    Constant used to signify a ‘drop’ verdict

br_msg
    Message age in 1/256 secs

nf_queue
    Constant used to signify a ‘queue’ verdict

br_mac
    Bridge MAC address

br_fd
    Forward delay in 1/256 secs

brhdr
    Address of bridge header

br_hptime
    Hello time in 1/256 secs

br_bid
    Identity of bridge

br_rmac
    Root bridge MAC address

protocol
    Packet protocol

br_prid
    Protocol identifier

br_type
    BPDU type
```
\textbf{nf\_stop}
\hspace{1em} Constant used to signify a 'stop' verdict

\textbf{br\_max}
\hspace{1em} Max age in 1/256 secs

\textbf{br\_rid}
\hspace{1em} Identity of root bridge

\textbf{br\_flags}
\hspace{1em} BPDU flags

\textbf{outdev\_name}
\hspace{1em} Name of network device packet will be routed to (if known)

\textbf{nf\_accept}
\hspace{1em} Constant used to signify an 'accept' verdict

\textbf{indev\_name}
\hspace{1em} Name of network device packet was received on (if known)

\textbf{br\_poid}
\hspace{1em} Port identifier

\textbf{outdev}
\hspace{1em} Address of net\_device representing output device, 0 if unknown

\textbf{nf\_repeat}
\hspace{1em} Constant used to signify a 'repeat' verdict

\textbf{br\_vid}
\hspace{1em} Protocol version identifier

\textbf{length}
\hspace{1em} The length of the packet buffer contents, in bytes

\textbf{nf\_stolen}
\hspace{1em} Constant used to signify a 'stolen' verdict

\textbf{br\_cost}
\hspace{1em} Total cost from transmitting bridge to root

\textbf{llcpdu}
\hspace{1em} Address of LLC Protocol Data Unit
```

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

SYNOPSIS

netfilter.bridge.pre_routing

VALUES

llcproto_stp
  Constant used to signify Bridge Spanning Tree Protocol packet

pf
  Protocol family -- always “bridge”

nf_drop
  Constant used to signify a ‘drop’ verdict

br_msg
  Message age in 1/256 secs

indev
  Address of net_device representing input device, 0 if unknown

brhdr
  Address of bridge header

nf_queue
  Constant used to signify a ‘queue’ verdict

br_fd
  Forward delay in 1/256 secs

br_mac
  Bridge MAC address

br_rid
  Identity of root bridge

nf_accept
  Constant used to signify an ‘accept’ verdict

br_flags
  BPDU flags

outdev_name
Name of network device packet will be routed to (if known)

**br_prid**
Protocol identifier

**br_rmac**
Root bridge MAC address

**br_hetime**
Hello time in 1/256 secs

**br_bid**
Identity of bridge

**protocol**
Packet protocol

**br_max**
Max age in 1/256 secs

**br_type**
BPDU type

**nf_stop**
Constant used to signify a 'stop' verdict

**br_cost**
Total cost from transmitting bridge to root

**nf_stolen**
Constant used to signify a 'stolen' verdict

**length**
The length of the packet buffer contents, in bytes

**llcpdu**
Address of LLC Protocol Data Unit

**outdev**
Address of net_device representing output device, 0 if unknown

**nf_repeat**
Constant used to signify a 'repeat' verdict

**indev_name**
Name of network device packet was received on (if known)

**br_poid**
Port identifier

**br_vid**
Protocol version identifier

---

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

**SYNOPSIS**

```
netfilter.ip.forward
```

**VALUES**

**saddr**
A string representing the source IP address

**sport**
TCP or UDP source port (ipv4 only)

**daddr**
A string representing the destination IP address

**pf**
Protocol family -- either “ipv4” or “ipv6”

**indev**
Address of net_device representing input device, 0 if unknown

**nf_drop**
Constant used to signify a 'drop' verdict

**nf_queue**
Constant used to signify a 'queue' verdict

**dport**
TCP or UDP destination port (ipv4 only)

**iphdr**
Address of IP header
**fin**
TCP FIN flag (if protocol is TCP; ipv4 only)

**ack**
TCP ACK flag (if protocol is TCP; ipv4 only)

**syn**
TCP SYN flag (if protocol is TCP; ipv4 only)

**ipproto_udp**
Constant used to signify that the packet protocol is UDP

**outdev_name**
Name of network device packet will be routed to (if known)

**nf_accept**
Constant used to signify an 'accept' verdict

**rst**
TCP RST flag (if protocol is TCP; ipv4 only)

**protocol**
Packet protocol from driver (ipv4 only)

**nf_stop**
Constant used to signify a 'stop' verdict

**length**
The length of the packet buffer contents, in bytes

**nf_stolen**
Constant used to signify a 'stolen' verdict

**urg**
TCP URG flag (if protocol is TCP; ipv4 only)

**psh**
TCP PSH flag (if protocol is TCP; ipv4 only)

**ipproto_tcp**
Constant used to signify that the packet protocol is TCP

**indev_name**
Name of network device packet was received on (if known)
**family**  
IP address family

**outdev**  
Address of net_device representing output device, 0 if unknown

**nf_repeat**  
Constant used to signify a 'repeat' verdict

---

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

**SYNOPSIS**  
```c
netfilter.ip.local_in
```

**VALUES**

**nf_stolen**  
Constant used to signify a 'stolen' verdict

**length**  
The length of the packet buffer contents, in bytes

**urg**  
TCP URG flag (if protocol is TCP; ipv4 only)

**psh**  
TCP PSH flag (if protocol is TCP; ipv4 only)

**nf_repeat**  
Constant used to signify a 'repeat' verdict

**family**  
IP address family

**outdev**  
Address of net_device representing output device, 0 if unknown

**ipproto_tcp**  
Constant used to signify that the packet protocol is TCP

**indev_name**  
Name of network device packet was received on (if known)
**nf_accept**
Constant used to signify an 'accept' verdict

**outdev_name**
Name of network device packet will be routed to (if known)

**protocol**
Packet protocol from driver (ipv4 only)

**rst**
TCP RST flag (if protocol is TCP; ipv4 only)

**nf_stop**
Constant used to signify a 'stop' verdict

**nf_queue**
Constant used to signify a 'queue' verdict

**dport**
TCP or UDP destination port (ipv4 only)

**iphdr**
Address of IP header

**fin**
TCP FIN flag (if protocol is TCP; ipv4 only)

**syn**
TCP SYN flag (if protocol is TCP; ipv4 only)

**ack**
TCP ACK flag (if protocol is TCP; ipv4 only)

**ipproto_udp**
Constant used to signify that the packet protocol is UDP

**saddr**
A string representing the source IP address

**sport**
TCP or UDP source port (ipv4 only)

**pf**
Protocol family -- either “ipv4” or “ipv6”
**NAME**

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

**SYNOPSIS**

```c
netfilter.ip.local_out
```

**VALUES**

**dport**

TCP or UDP destination port (ipv4 only)

**nf_queue**

Constant used to signify a 'queue' verdict

**syn**

TCP SYN flag (if protocol is TCP; ipv4 only)

**ipproto_udp**

Constant used to signify that the packet protocol is UDP

**ack**

TCP ACK flag (if protocol is TCP; ipv4 only)

**fin**

TCP FIN flag (if protocol is TCP; ipv4 only)

**iphdr**

Address of IP header

**saddr**

A string representing the source IP address

**sport**

TCP or UDP source port (ipv4 only)
indev
Address of net_device representing input device, 0 if unknown

nf_drop
Constant used to signify a 'drop' verdict

daddr
A string representing the destination IP address

pf
Protocol family -- either "ipv4" or "ipv6"

psh
TCP PSH flag (if protocol is TCP; ipv4 only)

urg
TCP URG flag (if protocol is TCP; ipv4 only)

length
The length of the packet buffer contents, in bytes

nf_stolen
Constant used to signify a 'stolen' verdict

ipproto_tcp
Constant used to signify that the packet protocol is TCP

indev_name
Name of network device packet was received on (if known)

nf_repeat
Constant used to signify a 'repeat' verdict

family
IP address family

outdev
Address of net_device representing output device, 0 if unknown

outdev_name
Name of network device packet will be routed to (if known)

nf_accept
Constant used to signify an 'accept' verdict
**nf_stop**
Constant used to signify a 'stop' verdict

**rst**
TCP RST flag (if protocol is TCP; ipv4 only)

**protocol**
Packet protocol from driver (ipv4 only)

---

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

**SYNOPSIS**

```
netfilter.ip.post_routing
```

**VALUES**

**family**
IP address family

**outdev**
Address of net_device representing output device, 0 if unknown

**nf_repeat**
Constant used to signify a 'repeat' verdict

**ipproto_tcp**
Constant used to signify that the packet protocol is TCP

**indev_name**
Name of network device packet was received on (if known)

**nf_stolen**
Constant used to signify a 'stolen' verdict

**length**
The length of the packet buffer contents, in bytes

**urg**
TCP URG flag (if protocol is TCP; ipv4 only)

**psh**
TCP PSH flag (if protocol is TCP; ipv4 only)
**rst**
TCP RST flag (if protocol is TCP; ipv4 only)

**protocol**
Packet protocol from driver (ipv4 only)

**nf_stop**
Constant used to signify a ‘stop’ verdict

**nf_accept**
Constant used to signify an 'accept' verdict

**outdev_name**
Name of network device packet will be routed to (if known)

**iphdr**
Address of IP header

**fin**
TCP FIN flag (if protocol is TCP; ipv4 only)

**syn**
TCP SYN flag (if protocol is TCP; ipv4 only)

**ipproto_udp**
Constant used to signify that the packet protocol is UDP

**ack**
TCP ACK flag (if protocol is TCP; ipv4 only)

**nf_queue**
Constant used to signify a ‘queue’ verdict

**dport**
TCP or UDP destination port (ipv4 only)

**pf**
Protocol family -- either “ipv4” or “ipv6”

**daddr**
A string representing the destination IP address

**nf_drop**
Constant used to signify a 'drop' verdict
\textbf{indev}
Address of net\_device representing input device, 0 if unknown

\textbf{saddr}
A string representing the source IP address

\textbf{sport}
TCP or UDP source port (ipv4 only)

\section*{NAME}
probe::netfilter.ip.pre\_routing — Called before an IP packet is routed

\section*{SYNOPSIS}

\begin{verbatim}
netfilter.ip.pre_routing
\end{verbatim}

\section*{VALUES}

\textbf{indev}
Address of net\_device representing input device, 0 if unknown

\textbf{nf\_drop}
Constant used to signify a 'drop' verdict

\textbf{daddr}
A string representing the destination IP address

\textbf{pf}
Protocol family - either 'ipv4' or 'ipv6'

\textbf{sport}
TCP or UDP source port (ipv4 only)

\textbf{saddr}
A string representing the source IP address

\textbf{syn}
TCP SYN flag (if protocol is TCP; ipv4 only)

\textbf{ipproto\_udp}
Constant used to signify that the packet protocol is UDP

\textbf{ack}
TCP ACK flag (if protocol is TCP; ipv4 only)
**iphdr**
Address of IP header

**fin**
TCP FIN flag (if protocol is TCP; ipv4 only)

**dport**
TCP or UDP destination port (ipv4 only)

**nf_queue**
Constant used to signify a 'queue' verdict

**nf_stop**
Constant used to signify a 'stop' verdict

**rst**
TCP RST flag (if protocol is TCP; ipv4 only)

**protocol**
Packet protocol from driver (ipv4 only)

**outdev_name**
Name of network device packet will be routed to (if known)

**nf_accept**
Constant used to signify an 'accept' verdict

**indev_name**
Name of network device packet was received on (if known)

**ipproto_tcp**
Constant used to signify that the packet protocol is TCP

**family**
IP address family

**nf_repeat**
Constant used to signify a 'repeat' verdict

**outdev**
Address of net_device representing output device, 0 if unknown

**psh**
TCP PSH flag (if protocol is TCP; ipv4 only)
TCP URG flag (if protocol is TCP; ipv4 only)

The length of the packet buffer contents, in bytes

Constant used to signify a ‘stolen’ verdict

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

SYNOPSIS

sunrpc.clnt.bind_new_program

VALUES

programe
the name of new RPC program

old_prog
the number of old RPC program

vers
the version of new RPC program

servername
the server machine name

old-vers
the version of old RPC program

old_programe
the name of old RPC program

progr
the number of new RPC program

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

sunrpc.clnt.call_async

VALUES

_progname_
    the RPC program name

_prot_
    the IP protocol number

_proc_
    the procedure number in this RPC call

_procname_
    the procedure name in this RPC call

_verse_
    the RPC program version number

_flags_
    flags

_servername_
    the server machine name

_xid_
    current transmission id

_port_
    the port number

_prog_
    the RPC program number

_dead_
    whether this client is abandoned

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

SYNOPSIS
sunrpc.clnt.call_sync

VALUES

xid
  current transmission id

servername
  the server machine name

flags
  flags

dead
  whether this client is abandoned

prog
  the RPC program number

port
  the port number

prot
  the IP protocol number

progname
  the RPC program name

vers
  the RPC program version number

proc
  the procedure number in this RPC call

procname
  the procedure name in this RPC call

NAME

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

SYNOPSIS

sunrpc.clnt.clone_client
VALUES

authflavor  
the authentication flavor

port  
the port number

programe  
the RPC program name

servername  
the server machine name

prot  
the IP protocol number

prog  
the RPC program number

vers  
the RPC program version number

NAME

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

SYNOPSIS

sunrpc.clnt.create_client

VALUES

servername  
the server machine name

prot  
the IP protocol number

authflavor  
the authentication flavor

port  
the port number
**NAME**
probe::sunrpc.clnt.restart_call — Restart an asynchronous RPC call

**SYNOPSIS**

```
sunrpc.clnt.restart_call
```

**VALUES**

- **servername**
  the server machine name

- **tk_priority**
  the task priority

- **xid**
  the transmission id

- **prog**
  the RPC program number

- **tk_runstate**
  the task run status

- **tk_pid**
  the debugging aid of task

- **tk_flags**
  the task flags

**NAME**
probe::sunrpc.clnt.shutdown_client — Shutdown an RPC client
SYNOPSIS

sunrpc.clnt.shutdown_client

VALUES

om_queue
the jiffies queued for xmit

clanes
the number of clones

vers
the RPC program version number

om_rtt
the RPC RTT jiffies

om_execute
the RPC execution jiffies

rpcctnt
the count of RPC calls

program
the RPC program name

authflavor
the authentication flavor

prot
the IP protocol number

prog
the RPC program number

om_bytes_recv
the count of bytes in

om_bytes_sent
the count of bytes out

port
the port number

om_ntrans
the count of RPC transmissions

netreconn
the count of reconnections

om_ops
the count of operations

tasks
the number of references

servername
the server machine name

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

SYNOPSIS

sunrpc.sched.delay

VALUES

prog
the program number in the RPC call

xid
the transmission id in the RPC call

delay
the time delayed

vers
the program version in the RPC call

tk_flags
the flags of the task

tk_pid
the debugging id of the task

prot
the IP protocol in the RPC call
NAME
probe::sunrpc.sched.execute — Execute the RPC `scheduler'

SYNOPSIS

sunrpc.sched.execute

VALUES

tk_pid
  the debugging id of the task

prot
  the IP protocol in the RPC call

vers
  the program version in the RPC call

tk_flags
  the flags of the task

xid
  the transmission id in the RPC call

prog
  the program number in the RPC call

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

SYNOPSIS

sunrpc.sched.new_task

VALUES

xid
  the transmission id in the RPC call

prog
  the program number in the RPC call

prot
the IP protocol in the RPC call

**vers**  
the program version in the RPC call

**tk_flags**  
the flags of the task

---

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

**SYNOPSIS**

```  
sunrpc.sched.release_task
```

**VALUES**

**prot**  
the IP protocol in the RPC call

**tk_flags**  
the flags of the task

**vers**  
the program version in the RPC call

**xid**  
the transmission id in the RPC call

**prog**  
the program number in the RPC call

**DESCRIPTION**

The `rpc_release_task` function might not be found for a particular kernel. So, if we can't find it, just return `-1` for everything.

---

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

**SYNOPSIS**

```  
sunrpc.svc.create
```
VALUES

bufsize
the buffer size

pg_nvers
the number of supported versions

progs
the name of the program

prog
the number of the program

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

SYNOPSIS

| sunrpc.svc.destroy |

VALUES

sv_nrthreads
the number of concurrent threads

sv_name
the service name

sv_prog
the number of the program

rpcbadauth
the count of requestsdropped for authentication failure

rpcbadfmt
the count of requests dropped for bad formats

rpccnt
the count of valid RPC requests

sv_progname
the name of the program
**NAME**
probe::sunrpc.svc.drop — Drop RPC request

**SYNOPSIS**

```
sunrpc.svc.drop
```

**VALUES**

- `rq_xid`:
  the transmission id in the request

- `sv_name`:
  the service name

- `rq_prot`:
  the IP protocol of the request

- `peer_ip`:
  the peer address where the request is from

- `rq_proc`:
  the procedure number in the request

- `rq_vers`:
  the program version in the request

- `rq_prog`:
  the program number in the request

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

**SYNOPSIS**

```
sunrpc.svc.process
```
VALUES

rq_prog
the program number in the request

rq_vers
the program version in the request

peer_ip
the peer address where the request is from

rq_proc
the procedure number in the request

sv_prog
the number of the program

rq_prot
the IP protocol of the request

sv_name
the service name

rq_xid
the transmission id in the request

sv_nbrthreads
the number of concurrent threads

NAME
probe::sunrpc.svc.recv — Listen for the next RPC request on any socket

SYNOPSIS

sunrpc.svc.recv

VALUES

sv_nbrthreads
the number of concurrent threads

sv_name
the service name
sv prog
   the number of the program

timeout
   the timeout of waiting for data

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

SYNOPSIS
sunrpc.svc.register

VALUES
sv_name
   the service name

prog
   the number of the program

port
   the port number

progsname
   the name of the program

prot
   the IP protocol number

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

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

SYNOPSIS
sunrpc.svc.send

VALUES
rq_vers
the program version in the request

rq_prog
the program number in the request

rq_prot
the IP protocol of the request

sv_name
the service name

rq_xid
the transmission id in the request

peer_ip
the peer address where the request is from

rq_proc
the procedure number in the request

NAME
probe::tcp.disconnect — TCP socket disconnection

SYNOPSIS

tcp.disconnect

VALUES

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

daddr
A string representing the destination IP address

sport
TCP source port

family
IP address family

name
Name of this probe
**saddr**
A string representing the source IP address

**dport**
TCP destination port

**sock**
Network socket

**CONTEXT**
The process which disconnects tcp

---

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

**SYNOPSIS**
- tcp.disconnect.return

**VALUES**

- **name**
  Name of this probe

- **ret**
  Error code (0: no error)

**CONTEXT**
The process which disconnects tcp

---

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

**SYNOPSIS**
- tcp.receive

**VALUES**

- **psh**
  TCP PSH flag

- **ack**
TCP ACK flag

daddr
A string representing the destination IP address

syn
TCP SYN flag

rst
TCP RST flag

sport
TCP source port

protocol
Packet protocol from driver

urg
TCP URG flag

name
Name of the probe point

family
IP address family

fin
TCP FIN flag

saddr
A string representing the source IP address

iphdr
IP header address

dport
TCP destination port

NAME
probe::tcp.recvmsg — Receiving TCP message

SYNOPSIS

tcp.recvmsg
VALUES

*daddr*
A string representing the destination IP address

*sport*
TCP source port

*size*
Number of bytes to be received

*name*
Name of this probe

*family*
IP address family

*saddr*
A string representing the source IP address

*sock*
Network socket

*dport*
TCP destination port

CONTEXT
The process which receives a tcp message

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

SYNOPSIS

| tcp.recvmsg.return

VALUES

*saddr*
A string representing the source IP address

*dport*
TCP destination port
**daddr**

A string representing the destination IP address

**size**

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

**sport**

TCP source port

**family**

IP address family

**name**

Name of this probe

---

**CONTEXT**

The process which receives a tcp message

---

**NAME**

probe::tcp.sendmsg — Sending a tcp message

**SYNOPSIS**

```
tcp.sendmsg
```

**VALUES**

**family**

IP address family

**sock**

Network socket

**name**

Name of this probe

**size**

Number of bytes to send

---

**CONTEXT**

The process which sends a tcp message
NAME
probe::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
probe::tcp.setsockopt — Call to setsockopt

SYNOPSIS

   tcp.setsockopt

VALUES

   optstr
       Resolves optname to a human-readable format

   name
       Name of this probe

   family
       IP address family

   level
       The level at which the socket options will be manipulated

   optname
       TCP socket options (e.g. TCP_NODELAY, TCP_MAXSEG, etc)

   sock
       Network socket
**optlen**

Used to access values for `setsockopt`

**CONTEXT**
The process which calls `setsockopt`

---

**NAME**

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

**SYNOPSIS**

```
tcp.setsockopt.return
```

**VALUES**

- `ret`
  
  Error code (0: no error)

- `name`
  
  Name of this probe

**CONTEXT**
The process which calls `setsockopt`

---

**NAME**

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

**SYNOPSIS**

```
udp.disconnect
```

**VALUES**

- `daddr`
  
  A string representing the destination IP address

- `sock`
  
  Network socket used by the process

- `saddr`
  
  A string representing the source IP address

- `sport`
UDP source port

flags
Flags (e.g. FIN, etc)

dport
UDP destination port

name
The name of this probe

family
IP address family

CONTEXT
The process which requests a UDP disconnection

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

SYNOPSIS
udp.disconnect.return

VALUES

saddr
A string representing the source IP address

sport
UDP source port

dport
UDP destination port

family
IP address family

name
The name of this probe

daddr
A string representing the destination IP address
**ERROR**

*Error code (0: no error)*

**CONTEXT**
The process which requested a UDP disconnection

---

**NAME**

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

**SYNOPSIS**

```
udp.recvmsg
```

**VALUES**

- `size`
  Number of bytes received by the process

- `sock`
  Network socket used by the process

- `daddr`
  A string representing the destination IP address

- `family`
  IP address family

- `name`
  The name of this probe

- `dport`
  UDP destination port

- `saddr`
  A string representing the source IP address

- `sport`
  UDP source port

**CONTEXT**
The process which received a UDP message
NAME
probe::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

family
  IP address family

dport
  UDP destination port

saddr
  A string representing the source IP address

sport
  UDP source port

size
  Number of bytes received by the process

daddr
  A string representing the destination IP address

CONTEXT
  The process which received a UDP message

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

SYNOPSIS
udp.sendmsg

VALUES

daddr
  A string representing the destination IP address
sock
  Network socket used by the process

size
  Number of bytes sent by the process

saddr
  A string representing the source IP address

sport
  UDP source port

family
  IP address family

name
  The name of this probe

dport
  UDP destination port

CONTEXT
The process which sent a UDP message

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

SYNOPSIS
  udp.sendmsg.return

VALUES
  size
    Number of bytes sent by the process

  name
    The name of this probe

CONTEXT
The process which sent a UDP message
CHAPTER 15. SOCKET TAPSET

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

**NAME**
function::inet_get_ip_source — Provide IP source address string for a kernel socket

**SYNOPSIS**

```
inet_get_ip_source:string(sock:long)
```

**ARGUMENTS**

```sock```

pointer to the kernel socket

**NAME**
function::inet_get_local_port — Provide local port number for a kernel socket

**SYNOPSIS**

```
inet_get_local_port:long(sock:long)
```

**ARGUMENTS**

```sock```

pointer to the kernel socket

**NAME**
function::sock_fam_num2str — Given a protocol family number, return a string representation

**SYNOPSIS**

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

**ARGUMENTS**

```family```

The family number
NAME
function::sock_fam_str2num — Given a protocol family name (string), return the corresponding protocol family number

SYNOPSIS
sock_fam_str2num:long(family:string)

ARGUMENTS
family
The family name

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

SYNOPSIS
sock_prot_num2str:string(proto:long)

ARGUMENTS
proto
The protocol number

NAME
function::sock_prot_str2num — Given a protocol name (string), return the corresponding protocol number

SYNOPSIS
sock_prot_str2num:long(proto:string)

ARGUMENTS
proto
The protocol name

NAME
function::sock_state_num2str — Given a socket state number, return a string representation
SYNOPSIS

sock_state_num2str:string(state:long)

ARGUMENTS

state
    The state number

NAME

function::sock_state_str2num — Given a socket state string, return the corresponding state number

SYNOPSIS

sock_state_str2num:long(state:string)

ARGUMENTS

state
    The state name

NAME

probe::socket.aio_read — Receiving message via sock_aio_read

SYNOPSIS

socket.aio_read

VALUES

flags
    Socket flags value

type
    Socket type value

size
    Message size in bytes

family
    Protocol family value

name
NAME
probe::socket.aio_read.return — Conclusion of message received via sock_aio_read

SYNOPSIS
socket.aio_read.return

VALUES

family
Protocol family value

protocol
Protocol value

name
Name of this probe

state
Socket state value

success
Was receive successful? (1 = yes, 0 = no)

flags
Socket flags value

type
Socket type value

size
Size of message received (in bytes) or error code if success = 0

**CONTEXT**
The message receiver.

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

**NAME**
probe::socket.aio_write — Message send via `sock_aio_write`

**SYNOPSIS**

```
socket.aio_write
```

**VALUES**

- **flags**
  
  Socket flags value

- **type**
  
  Socket type value

- **size**
  
  Message size in bytes

- **family**
  
  Protocol family value

- **protocol**
  
  Protocol value

- **name**
  
  Name of this probe

- **state**
  
  Socket state 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

socket.aio_write.return

VALUES

state
Socket state value

success
Was receive successful? (1 = yes, 0 = no)

family
Protocol family value

protocol
Protocol value

name
Name of this probe

type
Socket type value

size
Size of message received (in bytes) or error code if success = 0

flags
Socket flags 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.close — Close a socket

SYNOPSIS

socket.close
VALUES

*type*
Socket type value

*flags*
Socket flags value

*state*
Socket state value

*family*
Protocol family value

*name*
Name of this probe

*protocol*
Protocol 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

```
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
probe::socket.create — Creation of a socket

SYNOPSIS

socket.create

VALUES

type
   Socket type value

name
   Name of this probe

protocol
   Protocol value

family
   Protocol family value

requester
   Requested by user process or the kernel (1 = kernel, 0 = user)

CONTEXT
The requester (see requester variable)

DESCRIPTION
Fires at the beginning of creating a socket.

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

SYNOPSIS

socket.create.return

VALUES

success
   Was socket creation successful? (1 = yes, 0 = no)

family
   Protocol family value
**requester**
Requested by user process or the kernel (1 = kernel, 0 = user)

**name**
Name of this probe

**protocol**
Protocol value

**type**
Socket type value

**err**
Error code if success == 0

**CONTEXT**
The requester (user process or kernel)

**DESCRIPTION**
Fires at the conclusion of creating a socket.

---

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

**SYNOPSIS**

```
socket.read_iter
```

**VALUES**

**state**
Socket state value

**protocol**
Protocol value

**name**
Name of this probe

**family**
Protocol family value

**size**
Message size in bytes
**type**
Socket type value

**flags**
Socket flags value

**CONTEXT**
The message sender

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

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

**SYNOPSIS**

```
socket.read_iter.return
```

**VALUES**

**flags**
Socket flags value

**type**
Socket type value

**size**
Size of message received (in bytes) or error code if success = 0

**family**
Protocol family value

**name**
Name of this probe

**protocol**
Protocol value

**state**
Socket state value

**success**
Was receive successful? (1 = yes, 0 = no)
The message receiver.

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

---

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

**SYNOPSIS**

```c
socket.readv
```

**VALUES**

- `state`
  Socket state value

- `family`
  Protocol family value

- `protocol`
  Protocol value

- `name`
  Name of this probe

- `type`
  Socket type value

- `size`
  Message size in bytes

- `flags`
  Socket flags value

---

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 \texttt{sock\_readv}

\textbf{SYNOPSIS}

\begin{itemize}
  \item \texttt{socket.readv.return}
\end{itemize}

\textbf{VALUES}

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

\textbf{CONTEXT}

The message receiver.

\textbf{DESCRIPTION}

Fires at the conclusion of receiving a message on a socket via the \texttt{sock\_readv} function.

\textbf{NAME}

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

\textbf{SYNOPSIS}

\begin{itemize}
  \item \texttt{socket.receive}
\end{itemize}
VALUES

name
   Name of this probe

protocol
   Protocol value

family
   Protocol family value

success
   Was send successful? (1 = yes, 0 = no)

state
   Socket state value

flags
   Socket flags value

size
   Size of message received (in bytes) or error code if success = 0

type
   Socket type value

CONTEXT
The message receiver

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

SYNOPSIS

socket.recvmsg

VALUES

family
   Protocol family value

name
   Name of this probe
**protocol**
Protocol value

**state**
Socket state value

**flags**
Socket flags value

**type**
Socket type value

**size**
Message size in bytes

**CONTEXT**
The message receiver.

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

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

**SYNOPSIS**

```plaintext
socket.recvmsg.return
```

**VALUES**

**family**
Protocol family value

**name**
Name of this probe

**protocol**
Protocol value

**state**
Socket state value

**success**
Was receive successful? (1 = yes, 0 = no)
flags
Socket flags value

type
Socket type value

size
Size of message received (in bytes) or error code if success = 0

CONTEXT
The message receiver.

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

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

SYNOPSIS

VALUES

flags
Socket flags value

size
Size of message sent (in bytes) or error code if success = 0

type
Socket type value

protocol
Protocol value

name
Name of this probe

family
Protocol family value

success
Was send successful? (1 = yes, 0 = no)
**state**

Socket state value

**CONTEXT**

The message sender

---

**NAME**

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

**SYNOPSIS**

```":socket.sendmsg```

**VALUES**

- **family**
  Protocol family value

- **name**
  Name of this probe

- **protocol**
  Protocol value

- **state**
  Socket state value

- **flags**
  Socket flags value

- **type**
  Socket type value

- **size**
  Message size in bytes

**CONTEXT**

The message sender

**DESCRIPTION**

Fires at the beginning of sending a message on a socket via the `sock_sendmsg` function
NAME
probe::socket.sendmsg.return — Return from socket.sendmsg.

SYNOPSIS

socket.sendmsg.return

VALUES

type
Socket type value

size
Size of message sent (in bytes) or error code if success = 0

flags
Socket flags value

state
Socket state value

success
Was send successful? (1 = yes, 0 = no)

family
Protocol family value

protocol
Protocol value

name
Name of this probe

CONTEXT
The message sender.

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

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

SYNOPSIS

socket.write_iter
VALUES

state
Socket state value

family
Protocol family value

protocol
Protocol value

name
Name of this probe

type
Socket type value

size
Message size in bytes

flags
Socket flags value

CONTEXT
The message sender

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

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

SYNOPSIS

socket.write_iter.return

VALUES

type
Socket type value

size
Size of message received (in bytes) or error code if success = 0

flags
Socket flags value

**state**
Socket state value

**success**
Was receive successful? (1 = yes, 0 = no)

**family**
Protocol family value

**protocol**
Protocol value

**name**
Name of this probe

**CONTEXT**
The message receiver.

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

**NAME**
`probe::socket.writev` — Message sent via `socket_writev`

**SYNOPSIS**
```
socket.writev
```

**VALUES**

**state**
Socket state value

**protocol**
Protocol value

**name**
Name of this probe

**family**
Protocol family value

**size**
Message size in bytes

**type**
Socket type value

**flags**
Socket flags 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**

```plaintext
socket.writev.return
```

**VALUES**

**success**
Was send successful? (1 = yes, 0 = no)

**state**
Socket state value

**name**
Name of this probe

**protocol**
Protocol value

**family**
Protocol family value

**size**
Size of message sent (in bytes) or error code if success = 0

**type**
Socket type value

**flags**
Socket flags value

**CONTEXT**
The message receiver.

**DESCRIPTION**
Fires at the conclusion of sending a message on a socket via the `sock_writev` function.
CHAPTER 16. SNMP INFORMATION TAPSET

This family of probe points is used to probe socket activities to provide SNMP type information. It contains the following functions and probe points:

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

SYNOPSIS

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

ARGUMENTS

- `skb`  
  pointer to the struct sk_buff

- `op`  
  value to be counted if `skb` passes the filter

- `SourceIsLocal`  
  1 is local operation and 0 is non-local operation

DESCRIPTION

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

NAME
function::ipmib_get_proto — Get the protocol value

SYNOPSIS

```
ipmib_get_proto:long(skb:long)
```

ARGUMENTS

- `skb`  
  pointer to a struct sk_buff

DESCRIPTION

Returns the protocol value from `skb`. 
NAME
c function::ipmib_local_addr — Get the local ip address

SYNOPSIS

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

ARGUMENTS

skb
pointer to a struct sk_buff

SourceIsLocal
flag to indicate whether local operation

DESCRIPTION
Returns the local ip address skb.

NAME
c function::ipmib_remote_addr — Get the remote ip address

SYNOPSIS

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

ARGUMENTS

skb
pointer to a struct sk_buff

SourceIsLocal
flag to indicate whether local operation

DESCRIPTION
Returns the remote ip address from skb.

NAME
c function::ipmib_tcp_local_port — Get the local tcp port

SYNOPSIS

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

skb
pointer to a struct sk_buff

SourceIsLocal
flag to indicate whether local operation

DESCRIPTION
Returns the local tcp port from skb.

NAME
function::ipmib_tcp_remote_port — Get the remote tcp port

SYNOPSIS

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

ARGUMENTS

skb
pointer to a struct sk_buff

SourceIsLocal
flag to indicate whether local operation

DESCRIPTION
Returns the remote tcp port from skb.

NAME
function::linuxmib_filter_key — Default filter function for linuxmib.* probes

SYNOPSIS

linuxmib_filter_key:long(sk:long,op:long)

ARGUMENTS

sk
pointer to the struct sock

op
value to be counted if sk passes the filter
DESCRIPTION
This function is a default filter function. The user can replace this function with their own. The user-supplied filter function returns an index key based on the values in \textit{sk}. A return value of 0 means this particular \textit{sk} should be not be counted.

NAME
function::tcpmib_filter_key — Default filter function for tcpmib.* probes

SYNOPSIS
\begin{verbatim}
tcpmib_filter_key:long(sk:long,op:long)
\end{verbatim}

ARGUMENTS
\begin{description}
\item[sk] pointer to the struct sock being acted on
\item[op] value to be counted if \textit{sk} passes the filter
\end{description}

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

NAME
function::tcpmib_get_state — Get a socket's state

SYNOPSIS
\begin{verbatim}
tcpmib_get_state:long(sk:long)
\end{verbatim}

ARGUMENTS
\begin{description}
\item[sk] pointer to a struct sock
\end{description}

DESCRIPTION
Returns the \textit{sk} state from a struct sock.

NAME
function::tcpmib_local_addr — Get the source address
SYNOPSIS

tcpmib_local_addr:long(sk:long)

ARGUMENTS

sk
  pointer to a struct inet_sock

DESCRIPTION

Returns the saddr from a struct inet_sock in host order.

NAME

function::tcpmib_local_port — Get the local port

SYNOPSIS

tcpmib_local_port:long(sk:long)

ARGUMENTS

sk
  pointer to a struct inet_sock

DESCRIPTION

Returns the sport from a struct inet_sock in host order.

NAME

function::tcpmib_remote_addr — Get the remote address

SYNOPSIS

tcpmib_remote_addr:long(sk:long)

ARGUMENTS

sk
  pointer to a struct inet_sock

DESCRIPTION

Returns the daddr from a struct inet_sock in host order.
NAME
function::tcpmib_remote_port — Get the remote port

SYNOPSIS

```
tcpmib_remote_port:long(sk:long)
```

ARGUMENTS

```
sk
  pointer to a struct inet_sock
```

DESCRIPTION

Returns the dport from a struct inet_sock in host order.

NAME
probe::ipmib.ForwDatagrams — Count forwarded packet

SYNOPSIS

```
ipmib.ForwDatagrams
```

VALUES

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

```
skb
  pointer to the struct sk_buff being acted on
```

DESCRIPTION

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

NAME
probe::ipmib.FragFails — Count datagram fragmented unsuccessfully

SYNOPSIS

```
ipmib.FragFails
```

VALUES

```
...
```
**DESCRIPTION**
The packet pointed to by `skb` is filtered by the function `ipmib_filter_key`. If the packet passes the filter, it is counted in the global `FragFails` (equivalent to SNMP's MIB IPSTATS_MIB_FRAGFAILS).

**NAME**
`probe::ipmib.FragOKs` — Count datagram fragmented successfully

**SYNOPSIS**

```plaintext
ipmib.FragOKs
```

**VALUES**

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

**DESCRIPTION**
The packet pointed to by `skb` is filtered by the function `ipmib_filter_key`. If the packet passes the filter, it is counted in the global `FragOKs` (equivalent to SNMP's MIB IPSTATS_MIB_FRAGOKS).

**NAME**
`probe::ipmib.InAddrErrors` — Count arriving packets with an incorrect address

**SYNOPSIS**

```plaintext
ipmib.InAddrErrors
```

**VALUES**

- `skb`
  - pointer to the struct `sk_buff` being acted on
- `op`
  - value to be added to the counter (default value of 1)
DESCRIPTION
The packet pointed to by skb is filtered by the function ipmib_filter_key. If the packet passes the filter is is counted in the global InAddrErrors (equivalent to SNMP’s MIB IPSTATS_MIB_INADDRERRORS).

NAME
probe::ipmib.InDiscards — Count discarded inbound packets

SYNOPSIS
ipmib.InDiscards

VALUES

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

skb
pointer to the struct sk_buff being acted on

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

NAME
probe::ipmib.InNoRoutes — Count an arriving packet with no matching socket

SYNOPSIS
ipmib.InNoRoutes

VALUES

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

skb
pointer to the struct sk_buff being acted on

DESCRIPTION
The packet pointed to by skb is filtered by the function ipmib_filter_key. If the packet passes the filter is is counted in the global InNoRoutes (equivalent to SNMP’s MIB IPSTATS_MIB_INNOROUTES).
NAME
probe::ipmib.InReceives — Count an arriving packet

SYNOPSIS
ipmib.InReceives

VALUES

skb
pointer to the struct sk_buff being acted on

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

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

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

SYNOPSIS
ipmib.InUnknownProtos

VALUES

skb
pointer to the struct sk_buff being acted on

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

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

NAME
probe::ipmib.OutRequests — Count a request to send a packet

SYNOPSIS


ipmib.OutRequests

VALUES

skb
pointer to the struct sk_buff being acted on

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

DESCRIPTION
The packet pointed to by skb is filtered by the function ipmib_filter_key. If the packet passes the filter is is counted in the global OutRequests (equivalent to SNMP's MIB IPSTATS_MIB_OUTREQUESTS)

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

SYNOPSIS

ipmib.ReasmReqds

VALUES

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

skb
pointer to the struct sk_buff being acted on

DESCRIPTION
The packet pointed to by skb is filtered by the function ipmib_filter_key. If the packet passes the filter is is counted in the global ReasmReqds (equivalent to SNMP's MIB IPSTATS_MIB_REASMREQDS)

NAME
probe::ipmib.ReasmTimeout — Count Reassembly Timeouts

SYNOPSIS

ipmib.ReasmTimeout

VALUES
**op**

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

**skb**

pointer to the struct sk_buff being acted on

**DESCRIPTION**

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

**NAME**

`probes linuxmib.DelayedACKs` — Count of delayed acks

**SYNOPSIS**

```
linuxmib.DelayedACKs
```

**VALUES**

**op**

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

**sk**

Pointer to the struct sock being acted on

**DESCRIPTION**

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

**NAME**

`probes linuxmib.ListenDrops` — Count of times conn request that were dropped

**SYNOPSIS**

```
linuxmib.ListenDrops
```

**VALUES**

**sk**

Pointer to the struct sock being acted on
**DESCRIPTION**
The packet pointed to by \texttt{skb} is filtered by the function \texttt{linuxmib\_filter\_key}. If the packet passes the filter is is counted in the global \texttt{ListenDrops} (equivalent to SNMP’s MIB LINUX\_MIB\_LISTENDROPS).

**NAME**
\texttt{probe::linuxmib.ListenOverflows} — Count of times a listen queue overflowed

**SYNOPSIS**

\begin{verbatim}
linuxmib.ListenOverflows
\end{verbatim}

**VALUES**

\begin{itemize}
\item \texttt{sk}
  Pointer to the struct sock being acted on
\item \texttt{op}
  Value to be added to the counter (default value of 1)
\end{itemize}

**DESCRIPTION**
The packet pointed to by \texttt{skb} is filtered by the function \texttt{linuxmib\_filter\_key}. If the packet passes the filter is is counted in the global \texttt{ListenOverflows} (equivalent to SNMP’s MIB LINUX\_MIB\_LISTENOVERFLOWS).

**NAME**
\texttt{probe::linuxmib.TCPMemoryPressures} — Count of times memory pressure was used

**SYNOPSIS**

\begin{verbatim}
linuxmib.TCPMemoryPressures
\end{verbatim}

**VALUES**

\begin{itemize}
\item \texttt{sk}
  Pointer to the struct sock being acted on
\item \texttt{op}
  Value to be added to the counter (default value of 1)
\end{itemize}
DESCRIPTION
The packet pointed to by skb is filtered by the function linuxmib_filter_key. If the packet passes the filter is is counted in the global TCPMemoryPressures (equivalent to SNMP’s MIB LINUX_MIB_TCPMEMORYPRESSURES)

NAME
tcpmib.ActiveOpens — Count an active opening of a socket

SYNOPSIS

tcpmib.ActiveOpens

VALUES

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

sk
pointer to the struct sock being acted on

DESCRIPTION
The packet pointed to by skb is filtered by the function tcpmib_filter_key. If the packet passes the filter is is counted in the global ActiveOpens (equivalent to SNMP’s MIB TCP_MIB_ACTIVEOPENS)

NAME
tcpmib.AttemptFails — Count a failed attempt to open a socket

SYNOPSIS

tcpmib.AttemptFails

VALUES

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

sk
pointer to the struct sock being acted on

DESCRIPTION
The packet pointed to by skb is filtered by the function tcpmib_filter_key. If the packet passes the filter is is counted in the global AttemptFails (equivalent to SNMP’s MIB TCP_MIB_ATTEMPTFAILS)
NAME
probe::tcpmib.CurrEstab — Update the count of open sockets

SYNOPSIS

```
tcpmib.CurrEstab
```

VALUES

```
sk
    pointer to the struct sock being acted on

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

DESCRIPTION
The packet pointed to by `skb` is filtered by the function `tcpmib_filter_key`. If the packet passes the filter is is counted in the global `CurrEstab` (equivalent to SNMP’s MIB TCP_MIB_CURRESTAB)

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

SYNOPSIS

```
tcpmib.EstabResets
```

VALUES

```
sk
    pointer to the struct sock being acted on

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

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

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

SYNOPSIS

```
tcpmib.InSegs
```
VALUES

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

sk
  pointer to the struct sock being acted on

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

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

SYNOPSIS

VALUES

sk
  pointer to the struct sock being acted on

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

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

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

SYNOPSIS

VALUES

sk
DESCRIPTION
The packet pointed to by skb is filtered by the function tcpmib_filter_key. If the packet passes the filter it is counted in the global OutSegs (equivalent to SNMP’s MIB TCP_MIB_OUTSEGS).

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

SYNOPSIS
| tcpmib.PassiveOpens

VALUES
| sk — pointer to the struct sock being acted on
| op — value to be added to the counter (default value of 1)

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

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

SYNOPSIS
| tcpmib.RetransSegs

VALUES
| op — value to be added to the counter (default value of 1)
| sk — pointer to the struct sock being acted on
DESCRIPTION
The packet pointed to by skb is filtered by the function tcpmib_filter_key. If the packet passes the filter is is counted in the global RetransSegs (equivalent to SNMP's MIB TCP_MIB_RETRANSSEGS)
CHAPTER 17. KERNEL PROCESS TAPSET

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

NAME
function::get_loadavg_index — Get the load average for a specified interval

SYNOPSIS

get_loadavg_index:long(index:long)

ARGUMENTS

index
The load average interval to capture.

DESCRIPTION

This function returns the load average at a specified interval. The three load average values 1, 5 and 15 minute average corresponds to indexes 0, 1 and 2 of the avenrun array - see linux/sched.h. Please note that the truncated-integer portion of the load average is returned. If the specified index is out-of-bounds, then an error message and exception is thrown.

NAME
function::sprint_loadavg — Report a pretty-printed load average

SYNOPSIS

sprint_loadavg:string()

ARGUMENTS

None

DESCRIPTION

Returns the a string with three decimal numbers in the usual format for 1-, 5- and 15-minute load averages.

NAME
function::target_set_pid — Does pid descend from target process?

SYNOPSIS

target_set_pid(pid:)


ARGUMENTS

pid
  The pid of the process to query

DESCRIPTION
This function returns whether the given process-id is within the “target set”, that is whether it is a descendant of the top-level target process.

NAME
function::target_set_report — Print a report about the target set

SYNOPSIS

| target_set_report()

ARGUMENTS
None

DESCRIPTION
This function prints a report about the processes in the target set, and their ancestry.

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

SYNOPSIS

| kprocess.create

VALUES

| new_tid
  The TID of the newly created task

| 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 fork (or one of its syscall variants), or a new kernel thread.
NAME
probe::kprocess.exec — Attempt to exec to a new program

SYNOPSIS
kprocess.exec

VALUES

`filename`
   The path to the new executable

`name`
   Name of the system call ("execve") (SystemTap v2.5+)

`args`
   The arguments to pass to the new executable, including the 0th arg (SystemTap v2.5+)

`argstr`
   A string containing the filename followed by the arguments to pass, excluding 0th arg (SystemTap v2.5+)

CONTEXT
The caller of exec.

DESCRIPTION
Fires whenever a process attempts to exec to a new program. Aliased to the syscall.execve probe in SystemTap v2.5+.

---

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

SYNOPSIS
kprocess.exec_complete

VALUES

`retstr`
   A string representation of errno (SystemTap v2.5+)

`success`
   A boolean indicating whether the exec was successful

`errno`
   The error number resulting from the exec
**name**

Name of the system call ("execve") (SystemTap v2.5+)

**CONTEXT**

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

**DESCRIPTION**

Fires at the completion of an exec call. Aliased to the syscall.execve.return probe in SystemTap v2.5+.

**NAME**

probe::kprocess.exit — Exit from process

**SYNOPSIS**

- kprocess.exit

**VALUES**

- **code**
  - The exit code of the process

**CONTEXT**

The process which is terminating.

**DESCRIPTION**

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

**NAME**

probe::kprocess.release — Process released

**SYNOPSIS**

- kprocess.release

**VALUES**

- **released_tid**
  - TID of the task being released

- **task**
  - A task handle to the process being released

- **released_pid**
PID of the process being released

pid
Same as released_pid for compatibility (deprecated)

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.

NAME
probe::kprocess.start — Starting new process

SYNOPSIS

kprocess.start

VALUES
None

CONTEXT
Newly created process.

DESCRIPTION
Fires immediately before a new process begins execution.
CHAPTER 18. SIGNAL TAPSET

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

NAME
function::get_sa_flags — Returns the numeric value of sa_flags

SYNOPSIS
get_sa_flags:long(act:long)

ARGUMENTS
act
address of the sigaction to query.

NAME
function::get_sa_handler — Returns the numeric value of sa_handler

SYNOPSIS
get_sa_handler:long(act:long)

ARGUMENTS
act
address of the sigaction to query.

NAME
function::is_sig_blocked — Returns 1 if the signal is currently blocked, or 0 if it is not

SYNOPSIS
is_sig_blocked:long(task:long,sig:long)

ARGUMENTS
task
address of the task_struct to query.

sig
the signal number to test.
NAME
function::sa_flags_str — Returns the string representation of sa_flags

SYNOPSIS

```
  sa_flags_str: string(sa_flags: long)
```

ARGUMENTS

```
sa_flags
    the set of flags to convert to string.
```

NAME
function::sa_handler_str — Returns the string representation of an sa_handler

SYNOPSIS

```
  sa_handler_str(handler:)
```

ARGUMENTS

```
handler
    the sa_handler to convert to string.
```

DESCRIPTION
Returns the string representation of an sa_handler. If it is not SIG_DFL, SIG_IGN or SIG_ERR, it will return the address of the handler.

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

SYNOPSIS

```
  signal_str(num:)
```

ARGUMENTS

```
num
    the signal number to convert to string.
```
NAME
game::signal_mask_str — Returns the string representation of a sigset

SYNOPSIS

sigset_mask_str:string(mask:long)

ARGUMENTS

_mask
the sigset to convert to string.

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

SYNOPSIS

signal.check_ignored

VALUES

sig_pid
The PID of the process receiving the signal

sig
The number of the signal

sig_name
A string representation of the signal

pid_name
Name of the process receiving the signal

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

SYNOPSIS

signal.check_ignored.return

VALUES

name
Name of the probe point

\textit{retstr}
\begin{itemize}
\item Return value as a string
\end{itemize}

\section*{NAME}
probe::signal.checkperm — Check being performed on a sent signal

\section*{SYNOPSIS}

\begin{itemize}
\item \texttt{signal.checkperm}
\end{itemize}

\section*{VALUES}

\textit{pid\_name}
\begin{itemize}
\item Name of the process receiving the signal
\end{itemize}

\textit{task}
\begin{itemize}
\item A task handle to the signal recipient
\end{itemize}

\textit{sig\_name}
\begin{itemize}
\item A string representation of the signal
\end{itemize}

\textit{sinfo}
\begin{itemize}
\item The address of the siginfo structure
\end{itemize}

\textit{name}
\begin{itemize}
\item Name of the probe point
\end{itemize}

\textit{sig}
\begin{itemize}
\item The number of the signal
\end{itemize}

\textit{si\_code}
\begin{itemize}
\item Indicates the signal type
\end{itemize}

\textit{sig\_pid}
\begin{itemize}
\item The PID of the process receiving the signal
\end{itemize}

\section*{NAME}
probe::signal.checkperm.return — Check performed on a sent signal completed

\section*{SYNOPSIS}
VALUES

*retstr*

Return value as a string

*name*

Name of the probe point

---

NAME

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

SYNOPSIS

```
signal.do_action
```

VALUES

*sigact_addr*

The address of the new sigaction struct associated with the signal

*sig_name*

A string representation of the signal

*sa_mask*

The new mask of the signal

*sa_handler*

The new handler of the signal

*oldsigact_addr*

The address of the old sigaction struct associated with the signal

*sig*

The signal to be examined/changed

*name*

Name of the probe point

---

NAME

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

signal.do_action.return

VALUES

retstr
Return value as a string

name
Name of the probe point

NAME

probe::signal.flush — Flushing 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

name
Name of the probe point

sig_pid
The PID of the process associated with the task performing the flush

NAME

probe::signal.force_segv — Forcing send of SIGSEGV

SYNOPSIS

signal.force_segv

VALUES

sig_name
A string representation of the signal

**pid_name**
Name of the process receiving the signal

**sig_pid**
The PID of the process receiving the signal

**name**
Name of the probe point

**sig**
The number of the signal

---

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

**SYNOPSIS**

```
signal.force_segv.return
```

**VALUES**

**retstr**
Return value as a string

**name**
Name of the probe point

---

**NAME**
probe::signal.handle — Signal handler being invoked

**SYNOPSIS**

```
signal.handle
```

**VALUES**

**name**
Name of the probe point

**sig**
The signal number that invoked the signal handler

\textit{sinfo}

The address of the siginfo table

\textit{ka_addr}

The address of the k_sigaction table associated with the signal

\textit{sig_mode}

Indicates whether the signal was a user-mode or kernel-mode signal

\textit{sig_code}

The si_code value of the siginfo signal

\textit{regs}

The address of the kernel-mode stack area (deprecated in SystemTap 2.1)

\textit{oldset_addr}

The address of the bitmask array of blocked signals (deprecated in SystemTap 2.1)

\textit{sig_name}

A string representation of the signal

\begin{itemize}
  \item \textbf{NAME}
  \item \textbf{SYNOPSIS}
  \item \textbf{VALUES}
  \item \textbf{DESCRIPTION}
\end{itemize}

\textbf{NAME}

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

\textbf{SYNOPSIS}

\textbf{VALUES}

\textbf{DESCRIPTION}

(deprecated in SystemTap 2.1)
probe::signal.pending — Examining pending signal

**SYNOPSIS**

```
signal.pending
```

**VALUES**

- **name**
  - Name of the probe point

- **sigset_size**
  - The size of the user-space signal set

- **sigset_add**
  - The address of the user-space signal set (sigset_t)

**DESCRIPTION**

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

---

**NAME**

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

**SYNOPSIS**

```
signal.pending.return
```

**VALUES**

- **name**
  - Name of the probe point

- **retstr**
  - Return value as a string

---

**NAME**

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

**SYNOPSIS**

```
signal.procmask
```
VALUES

name
   Name of the probe point

sigset
   The actual value to be set for sigset_t (correct?)

how
   Indicates how to change the blocked signals; possible values are SIG_BLOCK=0 (for blocking signals), SIG_UNBLOCK=1 (for unblocking signals), and SIG_SETMASK=2 for setting the signal mask.

sigset_addr
   The address of the signal set (sigset_t) to be implemented

oldsigset_addr
   The old address of the signal set (sigset_t)

NAME

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

SYNOPSIS

signal.procmask.return

VALUES

retstr
   Return value as a string

name
   Name of the probe point

NAME

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

SYNOPSIS

signal.send

VALUES
**send2queue**

Indicates whether the signal is sent to an existing sigqueue (deprecated in SystemTap 2.1)

**pid_name**

The name of the signal recipient

**task**

A task handle to the signal recipient

**sig_name**

A string representation of the signal

**sinfo**

The address of siginfo struct

**shared**

Indicates whether the signal is shared by the thread group

**si_code**

Indicates the signal type

**name**

The name of the function used to send out the signal

**sig**

The number of the signal

**sig_pid**

The PID of the process receiving the signal

**CONTEXT**

The signal's sender.

**NAME**

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

**SYNOPSIS**

```
signal.send.return
```

**VALUES**

**shared**

Indicates whether the sent signal is shared by the thread group.
**name**

The name of the function used to send out the signal

**retstr**

The return value to either \_\_\_group\_\_send\_\_sig\_\_info, specific\_\_send\_\_sig\_\_info, or send\_\_sig\_\_queue

**send2queue**

Indicates whether the sent signal was sent to an existing sigqueue

**CONTEXT**

The signal's sender. (correct?)

**DESCRIPTION**

Possible \_\_\_group\_\_send\_\_sig\_\_info and specific\_\_send\_\_sig\_\_info return values are as follows;

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

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

Possible send\_\_group\_\_sig\_\_queue and send\_\_sig\_\_queue return values are as follows;

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

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

-1 -- (send\_\_sig\_\_queue only) The task was marked exiting, allowing * posix\_\_timer\_\_event to redirect it to the group leader.

**NAME**

probe::signal.send\_\_sig\_\_queue — Queuing a signal to a process

**SYNOPSIS**

\[
\texttt{signal.send\_\_sig\_\_queue}
\]

**VALUES**

**sig**

The queued signal

**name**

Name of the probe point

**sig\_\_pid**
The PID of the process to which the signal is queued

**pid_name**
Name of the process to which the signal is queued

**sig_name**
A string representation of the signal

**sigqueue_addr**
The address of the signal queue

---

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

**SYNOPSIS**

```markdown
signal.send_sig_queue.return
```

**VALUES**

**retstr**
Return value as a string

**name**
Name of the probe point

---

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

**SYNOPSIS**

```markdown
signal.sys_tgkill
```

**VALUES**

**sig_pid**
The PID of the thread receiving the kill signal

**sig**
The specific kill signal sent to the process

**name**
Name of the probe point

*pid_name*

The name of the signal recipient

*sig_name*

A string representation of the signal

*tgid*

The thread group ID of the thread receiving the kill signal

*task*

A task handle to the signal recipient

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

**NAME**

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

**SYNOPSIS**

```
signal.sys_tgkill.return
```

**VALUES**

*name*

Name of the probe point

*retstr*

The return value to either `__group_send_sig_info`,

**NAME**

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

**SYNOPSIS**

```
signal.sys_tkill
```

**VALUES**

*sig_pid*
The PID of the process receiving the kill signal

`sig`

The specific signal sent to the process

`name`

Name of the probe point

`pid_name`

The name of the signal recipient

`sig_name`

A string representation of the signal

`task`

A task handle to the signal recipient

**DESCRIPTION**

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

**NAME**

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

**SYNOPSIS**

```
signal.syskill
```

**VALUES**

`sig_pid`

The PID of the process receiving the signal

`sig`

The specific signal sent to the process

`name`

Name of the probe point

`pid_name`

The name of the signal recipient

`sig_name`

A string representation of the signal
**task**

A task handle to the signal recipient

---

**NAME**

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

**SYNOPSIS**

```plaintext
signal.syskill.return
```

**VALUES**

None

---

**NAME**

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

**SYNOPSIS**

```plaintext
signal.systkill.return
```

**VALUES**

*retstr*

The return value to either __group_send_sig_info,

*name*

Name of the probe point

---

**NAME**

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

**SYNOPSIS**

```plaintext
signal.wakeup
```

**VALUES**

*pid_name*

Name of the process to wake

*resume*
Indicates whether to wake up a task in a STOPPED or TRACED state

**state_mask**

A string representation indicating the mask of task states to wake. Possible values are TASK_INTERRUPTIBLE, TASK_STOPPED, TASK_TRACED, TASK_WAKEKILL, and TASK_INTERRUPTIBLE.

**sig_pid**

The PID of the process to wake
CHAPTER 19. ERRNO TAPSET

This set of functions is used to handle errno number values. It contains the following functions:

NAME

function::errno_str — Symbolic string associated with error code

SYNOPSIS

errno_str:string(err:long)

ARGUMENTS

err

The error number received

DESCRIPTION

This function returns the symbolic string associated with the giver error code, such as ENOENT for the number 2, or E#3333 for an out-of-range value such as 3333.

NAME

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

SYNOPSIS

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

ARGUMENTS

format

Variable to determine return type base value

ret

Return value (typically $return)

DESCRIPTION

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

Note that this function is preferred over returnstr.

NAME

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

returnstr:string(format:long)

ARGUMENTS

format
  Variable to determine return type base value

DESCRIPTION

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

Note that this function should only be used in dwarfless probes (i.e. "kprobe.function(“foo”)”). Other probes should use return_str.

NAME

function::returnval — Possible return value of probed function

SYNOPSIS

returnval:long()

ARGUMENTS

None

DESCRIPTION

Return the value of the register in which function values are typically returned. Can be used in probes where $return isn’t available. This is only a guess of the actual return value and can be totally wrong. Normally only used in dwarfless probes.
CHAPTER 20. RLIMIT TAPSET

This set of functions is used to handle string which defines resource limits (RLIMIT_*) and returns corresponding number of resource limit. It contains the following functions:

NAME
function::rlimit_from_str — Symbolic string associated with resource limit code

SYNOPSIS

```plaintext
rlimit_from_str:long(lim_str:string)
```

ARGUMENTS

```plaintext
lim_str
The string representation of limit
```

DESCRIPTION

This function returns the number associated with the given string, such as 0 for the string RLIMIT_CPU, or -1 for an out-of-range value.
CHAPTER 21. DEVICE TAPSET

This set of functions is used to handle kernel and userspace device numbers. It contains the following functions:

NAME
function::MAJOR — Extract major device number from a kernel device number (kdev_t)

SYNOPSIS

ARGUMENTS

NAME
function::MINOR — Extract minor device number from a kernel device number (kdev_t)

SYNOPSIS

ARGUMENTS

NAME
function::MKDEV — Creates a value that can be compared to a kernel device number (kdev_t)

SYNOPSIS

ARGUMENTS

major

Intended major device number.

minor

Intended minor device number.
NAME
function::usrdev2kerndev — Converts a user-space device number into the format used in the kernel

SYNOPSIS

usrdev2kerndev::long(dev:long)

ARGUMENTS

dev
    Device number in user-space format.
CHAPTER 22. DIRECTORY-ENTRY (DENTRY) TAPSET

This family of functions is used to map kernel VFS directory entry pointers to file or full path names.

NAME
function::d_name — get the dirent name

SYNOPSIS

d_name:string(dentry:long)

ARGUMENTS

dentry
    Pointer to dentry.

DESCRIPTION
Returns the dirent name (path basename).

NAME
function::d_path — get the full nameidata path

SYNOPSIS

d_path:string(nd:long)

ARGUMENTS

nd
    Pointer to nameidata.

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

NAME
function::fullpath_struct_file — get the full path

SYNOPSIS

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

ARGUMENTS

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

NAME
function::fullpath_struct_nameidata — get the full nameidata path

SYNOPSIS
fullpath_struct_nameidata(nd:)

ARGUMENTS
nd
Pointer to "struct nameidata".

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

NAME
function::fullpath_struct_path — get the full path

SYNOPSIS
fullpath_struct_path:string(path:long)

ARGUMENTS
path
Pointer to "struct path".

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

NAME
function::inode_name — get the inode name

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

inode_name:string(inode:long)

ARGUMENTS
inode
Pointer to inode.

DESCRIPTION
Returns the first path basename associated with the given inode.

NAME
function::inode_path — get the path to an inode

SYNOPSIS

inode_path:string(inode:long)

ARGUMENTS
inode
Pointer to inode.

DESCRIPTION
Returns the full path associated with the given inode.

NAME
function::real_mount — get the 'struct mount' pointer

SYNOPSIS

real_mount:long(vfsmnt:long)

ARGUMENTS
vfsmnt
Pointer to 'struct vfsmount'

DESCRIPTION
Returns the 'struct mount' pointer value for a 'struct vfsmount' pointer.
NAME
dunction::reverse_path_walk — get the full dirent path

SYNOPSIS

reverse_path_walk:string(dentry:long)

ARGUMENTS
dentry
    Pointer to dentry.

DESCRIPTION
Returns the path name (partial path to mount point).

NAME
dunction::task_dentry_path — get the full dentry path

SYNOPSIS

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

ARGUMENTS
task
    task_struct pointer.
dentry
    direntry pointer.
vfsmnt
    vfsmnt pointer.

DESCRIPTION
Returns the full dirent name (full path to the root), like the kernel d_path function.
CHAPTER 23. LOGGING TAPSET

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

NAME
function::assert — evaluate assertion

SYNOPSIS

```
assert(expression:,msg:)
```

ARGUMENTS

*expression*
  
The expression to evaluate

*msg*
  
The formatted message string

DESCRIPTION

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

NAME
function::error — Send an error message

SYNOPSIS

```
error(msg:string)
```

ARGUMENTS

*msg*
  
The formatted message string

DESCRIPTION

An implicit end-of-line is added. staprun prepends the string “ERROR:”. Sending an error message aborts the currently running probe. Depending on the MAXERRORS parameter, it may trigger an `exit`.

NAME
function::exit — Start shutting down probing script.

SYNOPSIS
exit()

ARGUMENTS
None

DESCRIPTION
This only enqueues a request to start shutting down the script. New probes will not fire (except “end” probes), but all currently running ones may complete their work.

NAME
function::ftrace — Send a message to the ftrace ring-buffer

SYNOPSIS
ftrace(msg:string)

ARGUMENTS
msg
  The formatted message string

DESCRIPTION
If the ftrace ring-buffer is configured & available, see /debugfs/tracing/trace for the message. Otherwise, the message may be quietly dropped. An implicit end-of-line is added.

NAME
function::log — Send a line to the common trace buffer

SYNOPSIS
log(msg:string)

ARGUMENTS
msg
  The formatted message string

DESCRIPTION
This function logs data. log sends the message immediately to staprun and to the bulk transport (relayfs) if it is being used. If the last character given is not a newline, then one is added. This function is not as efficient as printf and should be used only for urgent messages.
NAME
function::printk — Send a message to the kernel trace buffer

SYNOPSIS

printk(level:long,msg:string)

ARGUMENTS

level
an integer for the severity level (0=KERN_EMERG ... 7=KERN_DEBUG)

msg
The formatted message string

DESCRIPTION
Print a line of text to the kernel dmesg/console with the given severity. An implicit end-of-line is added. This function may not be safely called from all kernel probe contexts, so is restricted to guru mode only.

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

SYNOPSIS

warn(msg:string)

ARGUMENTS

msg
The formatted message string

DESCRIPTION
This function sends a warning message immediately to staprun. It is also sent over the bulk transport (relayfs) if it is being used. If the last character is not a newline, the one is added.
CHAPTER 24. QUEUE STATISTICS TAPSET

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

NAME
function::qs_done — Function to record finishing request

SYNOPSIS

qs_done(qname:string)

ARGUMENTS

qname
the name of the service that finished

DESCRIPTION
This function records that a request originally from the given queue has completed being serviced.

NAME
function::qs_run — Function to record being moved from wait queue to being serviced

SYNOPSIS

qs_run(qname:string)

ARGUMENTS

qname
the name of the service being moved and started

DESCRIPTION
This function records that the previous enqueued request was removed from the given wait queue and is now being serviced.

NAME
function::qs_wait — Function to record enqueue requests

SYNOPSIS

qs_wait(qname:string)

ARGUMENTS
**DESCRIPTION**
This function records that a new request was enqueued for the given queue name.

**NAME**
function::qsq_blocked — Returns the time request was on the wait queue

**SYNOPSIS**
```plaintext
qsq_blocked:long(qname:string,scale:long)
```

**ARGUMENTS**

- **qname**
  queue name

- **scale**
  scale variable to take account for interval fraction

**DESCRIPTION**
This function returns the fraction of elapsed time during which one or more requests were on the wait queue.

**NAME**
function::qsq_print — Prints a line of statistics for the given queue

**SYNOPSIS**
```plaintext
qsq_print(qname:string)
```

**ARGUMENTS**

- **qname**
  queue name

**DESCRIPTION**
This function prints a line containing the following

**STATISTICS FOR THE GIVEN QUEUE**
the queue name, the average rate of requests per second, the average wait queue length, the average
time on the wait queue, the average time to service a request, the percentage of time the wait queue
was used, and the percentage of time request was being serviced.

NAME
function::qsq_service_time — Amount of time per request service

SYNOPSIS

\[ \text{qsq\_service\_time}: \text{long}(\text{qname: string}, \text{scale: long}) \]

ARGUMENTS

**qname**

queue name

**scale**

scale variable to take account for interval fraction

DESCRIPTION

This function returns the average time in microseconds required to service a request once it is removed
from the wait queue.

NAME

function::qsq_start — Function to reset the stats for a queue

SYNOPSIS

\[ \text{qsq\_start}(\text{qname: string}) \]

ARGUMENTS

**qname**

the name of the service that finished

DESCRIPTION

This function resets the statistics counters for the given queue, and restarts tracking from the moment
the function was called. This function is also used to create intialize a queue.

NAME

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

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

ARGUMENTS

qname
    queue name

scale
    scale variable to take account for interval fraction

DESCRIPTION
This function returns the average number or requests served per microsecond.

NAME
function::qsq_utilization — Fraction of time that any request was being serviced

SYNOPSIS

qsq_utilization:long(qname:string,scale:long)

ARGUMENTS

qname
    queue name

scale
    scale variable to take account for interval fraction

DESCRIPTION
This function returns the average time in microseconds that at least one request was being serviced.

NAME
function::qsq_wait_queue_length — length of wait queue

SYNOPSIS

qsq_wait_queue_length:long(qname:string,scale:long)

ARGUMENTS

qname

queue name

\textit{scale}

scale variable to take account for interval fraction

**DESCRIPTION**
This function returns the average length of the wait queue

**NAME**
function::\texttt{qsq\_wait\_time} — Amount of time in queue + service per request

**SYNOPSIS**

\begin{verbatim}
qsq\_wait\_time:long(qname:string, scale:long)
\end{verbatim}

**ARGUMENTS**

\textit{qname}

queue name

\textit{scale}

scale variable to take account for interval fraction

**DESCRIPTION**
This function returns the average time in microseconds that it took for a request to be serviced (qs\_wait to qa\_done).
CHAPTER 25. RANDOM FUNCTIONS TAPSET

These functions deal with random number generation.

NAME
function::randint — Return a random number between [0,n)

SYNOPSIS

| randint:long(n:long) |

ARGUMENTS

n

Number past upper limit of range, not larger than 2**20.
CHAPTER 26. STRING AND DATA RETRIEVING FUNCTIONS

TAPSET

Functions to retrieve strings and other primitive types from the kernel or a user space programs based on addresses. All strings are of a maximum length given by MAXSTRINGLEN.

NAME
function::atomic_long_read — Retrieves an atomic long variable from kernel memory

SYNOPSIS

atomic_long_read:long(addr:long)

ARGUMENTS

addr
pointer to atomic long variable

DESCRIPTION
Safely perform the read of an atomic long variable. This will be a NOP on kernels that do not have ATOMIC_LONG_INIT set on the kernel config.

NAME
function::atomic_read — Retrieves an atomic variable from kernel memory

SYNOPSIS

atomic_read:long(addr:long)

ARGUMENTS

addr
pointer to atomic variable

DESCRIPTION
Safely perform the read of an atomic variable.

NAME
function::kernel_char — Retrieves a char value stored in kernel memory

SYNOPSIS

kernel_char:long(addr:long)
ARGUMENTS

addr

The kernel address to retrieve the char from

DESCRIPTION
Returns the char value from a given kernel memory address. Reports an error when reading from the given address fails.

NAME
function::kernel_int — Retrieves an int value stored in kernel memory

SYNOPSIS

kernel_int:long(addr:long)

ARGUMENTS

addr

The kernel address to retrieve the int from

DESCRIPTION
Returns the int value from a given kernel memory address. Reports an error when reading from the given address fails.

NAME
function::kernel_long — Retrieves a long value stored in kernel memory

SYNOPSIS

kernel_long:long(addr:long)

ARGUMENTS

addr

The kernel address to retrieve the long from

DESCRIPTION
Returns the long value from a given kernel memory address. Reports an error when reading from the given address fails.
NAME
function::kernel_pointer — Retrieves a pointer value stored in kernel memory

SYNOPSIS

```
kernel_pointer:long(addr:long)
```

ARGUMENTS

```
addr
  The kernel address to retrieve the pointer from
```

DESCRIPTION
Returns the pointer value from a given kernel memory address. Reports an error when reading from the
given address fails.

---

NAME
function::kernel_short — Retrieves a short value stored in kernel memory

SYNOPSIS

```
kernel_short:long(addr:long)
```

ARGUMENTS

```
addr
  The kernel address to retrieve the short from
```

DESCRIPTION
Returns the short value from a given kernel memory address. Reports an error when reading from the
given address fails.

---

NAME
function::kernel_string — Retrieves string from kernel memory

SYNOPSIS

```
kernel_string:string(addr:long)
```

ARGUMENTS

```
addr
  The kernel address to retrieve the string from
```
DESCRIPTION
This function returns the null terminated C string from a given kernel memory address. Reports an error on string copy fault.

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

SYNOPSIS

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

ARGUMENTS

addr
The kernel address to retrieve the string from

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

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

NAME
function::kernel_string2_utf16 — Retrieves UTF-16 string from kernel memory with alternative error string

SYNOPSIS

kernel_string2_utf16:string(addr:long,err_msg:string)

ARGUMENTS

addr
The kernel address to retrieve the string from

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

DESCRIPTION
This function returns a null terminated UTF-8 string converted from the UTF-16 string at a given kernel memory address. Reports the given error message on string copy fault or conversion error.
NAME
function::kernel_string2_utf32 — Retrieves UTF-32 string from kernel memory with alternative error string

SYNOPSIS

\[ \text{kernel_string2_utf32:string}(\text{addr:long}, \text{err\_msg:string}) \]

ARGUMENTS

\textit{addr}
   The kernel address to retrieve the string from

\textit{err\_msg}
   The error message to return when data isn't available

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

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

SYNOPSIS

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

ARGUMENTS

\textit{addr}
   The kernel address to retrieve the string from

\textit{n}
   The maximum length of the string (if not null terminated)

DESCRIPTION
Returns the C string of a maximum given length from a given kernel memory address. Reports an error on string copy fault.

NAME
function::kernel_string_quoted — Retrieves and quotes string from kernel memory

SYNOPSIS
kernel_string_quoted:string(addr:long)

ARGUMENTS

addr

the kernel memory address to retrieve the string from

DESCRIPTION

Returns the null terminated C string from a given kernel memory address where any ASCII characters that are not printable are replaced by the corresponding escape sequence in the returned string. Note that the string will be surrounded by double quotes. If the kernel memory data is not accessible at the given address, the address itself is returned as a string, without double quotes.

NAME

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

SYNOPSIS

kernel_string_quoted_utf16:string(addr:long)

ARGUMENTS

addr

The kernel address to retrieve the string from

DESCRIPTION

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

NAME

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

SYNOPSIS

kernel_string_quoted_utf32:string(addr:long)

ARGUMENTS

addr

The kernel address to retrieve the string from

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

**NAME**

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

**SYNOPSIS**

```
kernel_string_utf16:string(addr:long)
```

**ARGUMENTS**

`addr`

The kernel address to retrieve the string from

**DESCRIPTION**

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

**NAME**

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

**SYNOPSIS**

```
kernel_string_utf32:string(addr:long)
```

**ARGUMENTS**

`addr`

The kernel address to retrieve the string from

**DESCRIPTION**

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

**NAME**

`function::user_char` — Retrieves a char value stored in user space

**SYNOPSIS**

```
user_char:long(addr:long)
```
ARGUMENTS

`addr`  
the user space address to retrieve the char from

DESCRIPTION

Returns the char value from a given user space address. Returns zero when user space data is not accessible.

NAME

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

SYNOPSIS

```
user_char_warn:long(addr:long)
```

ARGUMENTS

`addr`  
the user space address to retrieve the char from

DESCRIPTION

Returns the char value from a given user space address. Returns zero when user space and warns (but does not abort) about the failure.

NAME

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

SYNOPSIS

```
user_int:long(addr:long)
```

ARGUMENTS

`addr`  
the user space address to retrieve the int from

DESCRIPTION

Returns the int value from a given user space address. Returns zero when user space data is not accessible.
NAME
t-function::user_int16 — Retrieves a 16-bit integer value stored in user space

SYNOPSIS

ARGUMENTS

.addr

the user space address to retrieve the 16-bit integer from

DESCRIPTION

Returns the 16-bit integer value from a given user space address. Returns zero when user space data is not accessible.

NAME

t-function::user_int32 — Retrieves a 32-bit integer value stored in user space

SYNOPSIS

ARGUMENTS

.addr

the user space address to retrieve the 32-bit integer from

DESCRIPTION

Returns the 32-bit integer value from a given user space address. Returns zero when user space data is not accessible.

NAME

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

SYNOPSIS

ARGUMENTS

.addr

the user space address to retrieve the 64-bit integer from
DESCRIPTION
Returns the 64-bit integer value from a given user space address. Returns zero when user space data is not accessible.

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

SYNOPSIS

user_int8:long(addr:long)

ARGUMENTS

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

DESCRIPTION
Returns the 8-bit integer value from a given user space address. Returns zero when user space data is not accessible.

NAME
function::user_int_warn — Retrieves an int value stored in user space

SYNOPSIS

user_int_warn:long(addr:long)

ARGUMENTS

addr
the user space address to retrieve the int from

DESCRIPTION
Returns the int value from a given user space address. Returns zero when user space and warns (but does not abort) about the failure.

NAME
function::user_long — Retrieves a long value stored in user space

SYNOPSIS

user_long:long(addr:long)
ARGUMENTS

*addr*

the user space address to retrieve the long from

DESCRIPTION

Returns the long value from a given user space address. Returns zero when user space data is not accessible. Note that the size of the long depends on the architecture of the current user space task (for those architectures that support both 64/32 bit compat tasks).

NAME

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

SYNOPSIS

```
user_long_warn:long(addr:long)
```

ARGUMENTS

*addr*

the user space address to retrieve the long from

DESCRIPTION

Returns the long value from a given user space address. Returns zero when user space data is not accessible. Note that the size of the long depends on the architecture of the current user space task (for those architectures that support both 64/32 bit compat tasks).

NAME

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

SYNOPSIS

```
user_short:long(addr:long)
```

ARGUMENTS

*addr*

the user space address to retrieve the short from

DESCRIPTION

Returns the short value from a given user space address. Returns zero when user space data is not accessible.
NAME
function::user_short_warn — Retrieves a short value stored in user space

SYNOPSIS

    user_short_warn:long(addr:long)

ARGUMENTS

addr
    the user space address to retrieve the short from

DESCRIPTION
Returns the short value from a given user space address. Returns zero when user space and warns (but
does not abort) about the failure.

NAME
function::user_string — Retrieves string from user space

SYNOPSIS

    user_string:string(addr:long)

ARGUMENTS

addr
    the user space address to retrieve the string from

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

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

SYNOPSIS

    user_string2:string(addr:long, err_msg:string)

ARGUMENTS

addr
    the user space address to retrieve the string from
**DESCRIPTION**

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

**NAME**

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

**SYNOPSIS**

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

**ARGUMENTS**

`addr`

the user space address to retrieve the string from

`n`

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

`warn_msg`

the warning message to return when data isn't available

**DESCRIPTION**

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

**NAME**

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

**SYNOPSIS**

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

**ARGUMENTS**

`addr`

The user address to retrieve the string from

`err_msg`

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

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

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

**SYNOPSIS**

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

**ARGUMENTS**

*addr*
- The user address to retrieve the string from

*err_msg*
- The error message to return when data isn't available

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

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

**SYNOPSIS**

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

**ARGUMENTS**

*addr*
- the user space address to retrieve the string from

*warn_msg*
- the warning message to return when data isn't available

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

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

SYNOPSIS

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

ARGUMENTS

addr
  the user space address to retrieve the string from

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

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

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

SYNOPSIS

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

ARGUMENTS

addr
  the user space address to retrieve the string from

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

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

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

SYNOPSIS

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

ARGUMENTS

- **addr**
  - the user space address to retrieve the string from

- **inlen**
  - the maximum length of the string to read (if not null terminated)

- **outlen**
  - the maximum length of the output string

DESCRIPTION

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

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

SYNOPSIS

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

ARGUMENTS

- **addr**
  - the user space address to retrieve the string from

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

DESCRIPTION

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

NAME
function::user_string_n_warn — Retrieves string from user space

SYNOPSIS

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

ARGUMENTS

addr
    the user space address to retrieve the string from

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

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

NAME
function::user_string_quoted — Retrives and quotes string from user space

SYNOPSIS

user_string_quoted:string(addr:long)

ARGUMENTS

addr
    the user space address to retrieve the string from

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

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

user_string_quoted_utf16:string(addr:long)

ARGUMENTS

addr

The user address to retrieve the string from

DESCRIPTION

This function combines quoting as per \textit{string_quoted} and UTF-16 decoding as per \textit{user_string_utf16}.

NAME

function::user_string_quoted_utf16 — Quote given user UTF-32 string.

SYNOPSIS

user_string_quoted_utf32:string(addr:long)

ARGUMENTS

addr

The user address to retrieve the string from

DESCRIPTION

This function combines quoting as per \textit{string_quoted} and UTF-32 decoding as per \textit{user_string_utf32}.

NAME

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

SYNOPSIS

user_string_utf16:string(addr:long)

ARGUMENTS

addr

The user address to retrieve the string from

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

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

**SYNOPSIS**

```c
user_string_utf32:string(addr:long)
```

**ARGUMENTS**

`addr`

The user address to retrieve the string from

**DESCRIPTION**

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

**NAME**
function::user_string_warn — Retrieves string from user space

**SYNOPSIS**

```c
user_string_warn:string(addr:long)
```

**ARGUMENTS**

`addr`

The user space address to retrieve the string from

**DESCRIPTION**

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

**NAME**
function::user_uint16 — Retrieves an unsigned 16-bit integer value stored in user space

**SYNOPSIS**

```c
user_uint16:long(addr:long)
```
ARGUMENTS

*addr*
the user space address to retrieve the unsigned 16-bit integer from

DESCRIPTION
Returns the unsigned 16-bit integer value from a given user space address. Returns zero when user space data is not accessible.

NAME
*function::user_uint32* — Retrieves an unsigned 32-bit integer value stored in user space

SYNOPSIS

```
user_uint32:long(addr:long)
```

ARGUMENTS

*addr*
the user space address to retrieve the unsigned 32-bit integer from

DESCRIPTION
Returns the unsigned 32-bit integer value from a given user space address. Returns zero when user space data is not accessible.

NAME
*function::user_uint64* — Retrieves an unsigned 64-bit integer value stored in user space

SYNOPSIS

```
user_uint64:long(addr:long)
```

ARGUMENTS

*addr*
the user space address to retrieve the unsigned 64-bit integer from

DESCRIPTION
Returns the unsigned 64-bit integer value from a given user space address. Returns zero when user space data is not accessible.
NAME
function::user_uint8 — Retrieves an unsigned 8-bit integer value stored in user space

SYNOPSIS
user_uint8:long(addr:long)

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

DESCRIPTION
Returns the unsigned 8-bit integer value from a given user space address. Returns zero when user space data is not accessible.

NAME
function::user_ulong — Retrieves an unsigned long value stored in user space

SYNOPSIS
user_ulong:long(addr:long)

ARGUMENTS
addr
  the user space address to retrieve the unsigned long from

DESCRIPTION
Returns the unsigned long value from a given user space address. Returns zero when user space data is not accessible. Note that the size of the unsigned long depends on the architecture of the current user space task (for those architectures that support both 64/32 bit compat tasks).

NAME
function::user_ulong_warn — Retrieves an unsigned long value stored in user space

SYNOPSIS
user_ulong_warn:long(addr:long)

ARGUMENTS
addr
  the user space address to retrieve the unsigned long from
**DESCRIPTION**
Returns the unsigned long value from a given user space address. Returns zero when user space and warns (but does not abort) about the failure. Note that the size of the unsigned long depends on the architecture of the current user space task (for those architectures that support both 64/32 bit compatible tasks).

**NAME**
function::user_ushort — Retrieves an unsigned short value stored in user space

**SYNOPSIS**

```plaintext
user_ushort:long(addr:long)
```

**ARGUMENTS**

`addr`

the user space address to retrieve the unsigned short from

**DESCRIPTION**
Returns the unsigned short value from a given user space address. Returns zero when user space data is not accessible.

**NAME**
function::user_ushort_warn — Retrieves an unsigned short value stored in user space

**SYNOPSIS**

```plaintext
user_ushort_warn:long(addr:long)
```

**ARGUMENTS**

`addr`

the user space address to retrieve the unsigned short from

**DESCRIPTION**
Returns the unsigned short value from a given user space address. Returns zero when user space and warns (but does not abort) about the failure.
CHAPTER 27. STRING AND DATA WRITING FUNCTIONS TAPSET

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

NAME
function::set_kernel_char — Writes a char value to kernel memory

SYNOPSIS

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

ARGUMENTS

- `addr`
  
The kernel address to write the char to

- `val`
  
The char which is to be written

DESCRIPTION

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

NAME
function::set_kernel_int — Writes an int value to kernel memory

SYNOPSIS

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

ARGUMENTS

- `addr`
  
The kernel address to write the int to

- `val`
  
The int which is to be written

DESCRIPTION

Writes the int value to a given kernel memory address. Reports an error when writing to the given address fails. Requires the use of guru mode (-g).
**NAME**
function::set_kernel_long — Writes a long value to kernel memory

**SYNOPSIS**

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

**ARGUMENTS**

- **addr**
  The kernel address to write the long to

- **val**
  The long which is to be written

**DESCRIPTION**

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

---

**NAME**
function::set_kernel_pointer — Writes a pointer value to kernel memory.

**SYNOPSIS**

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

**ARGUMENTS**

- **addr**
  The kernel address to write the pointer to

- **val**
  The pointer which is to be written

**DESCRIPTION**

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

---

**NAME**
function::set_kernel_short — Writes a short value to kernel memory

**SYNOPSIS**

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

---
ARGUMENTS

addr
  The kernel address to write the short to

val
  The short which is to be written

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

NAME
function::set_kernel_string — Writes a string to kernel memory

SYNOPSIS

set_kernel_string(addr:long,val:string)

ARGUMENTS

addr
  The kernel address to write the string to

val
  The string which is to be written

DESCRIPTION
 Writes the given string to a given kernel memory address. Reports an error on string copy fault. Requires the use of guru mode (-g).

NAME
function::set_kernel_string_n — Writes a string of given length to kernel memory

SYNOPSIS

set_kernel_string_n(addr:long,n:long,val:string)

ARGUMENTS

addr
  The kernel address to write the string to
**n**  
The maximum length of the string

**val**  
The string which is to be written

**DESCRIPTION**  
Writes the given string up to a maximum given length to a given kernel memory address. Reports an error on string copy fault. Requires the use of guru mode (-g).
CHAPTER 28. GURU TAPSETS

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

NAME
function::mdelay — millisecond delay

SYNOPSIS
mdelay(ms:long)

ARGUMENTS
ms
Number of milliseconds to delay.

DESCRIPTION
This function inserts a multi-millisecond busy-delay into a probe handler. It requires guru mode.

NAME
function::panic — trigger a panic

SYNOPSIS
panic(msg:string)

ARGUMENTS
msg
message to pass to kernel's panic function

DESCRIPTION
This function triggers an immediate panic of the running kernel with a user-specified panic message. It requires guru mode.

NAME
function::raise — raise a signal in the current thread

SYNOPSIS
raise(signo:long)
ARGUMENTS

`signo`

signal number

DESCRIPTION
This function calls the kernel `send_sig` routine on the current thread, with the given raw unchecked signal number. It may raise an error if `send_sig` failed. It requires guru mode.

NAME
`function::udelay — microsecond delay`

SYNOPSIS

```
udelay(us:long)
```

ARGUMENTS

`us`

Number of microseconds to delay.

DESCRIPTION
This function inserts a multi-microsecond busy-delay into a probe handler. It requires guru mode.
CHAPTER 29. A COLLECTION OF STANDARD STRING FUNCTIONS

Functions to get the length, a substring, getting at individual characters, string searching, escaping, tokenizing, and converting strings to longs.

NAME
function::isdigit — Checks for a digit

SYNOPSIS

isdigit:long(str:string)

ARGUMENTS

str
string to check

DESCRIPTION
Checks for a digit (0 through 9) as the first character of a string. Returns non-zero if true, and a zero if false.

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

SYNOPSIS

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

ARGUMENTS

s1
string to search in

s2
substring to find

DESCRIPTION
This function returns 1 if string s1 contains s2, otherwise zero.

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

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

ARGUMENTS

prnt_str
  the string to search and replace in

srch_str
  the substring which is used to search in prnt_str string

rplc_str
  the substring which is used to replace srch_str

DESCRIPTION
This function returns the given string with substrings replaced.

NAME
function::string_quoted — Quotes a given string

SYNOPSIS

  string_quoted:string(str:string)

ARGUMENTS

str
  The kernel address to retrieve the string from

DESCRIPTION
Returns the quoted string version of the given string, with characters where any ASCII characters that
are not printable are replaced by the corresponding escape sequence in the returned string. Note that
the string will be surrounded by double quotes.

NAME
function::stringat — Returns the char at a given position in the string

SYNOPSIS

  stringat:long(str:string,pos:long)

ARGUMENTS
The function returns the character at a given position in the string or zero if the string doesn't have as many characters. Reports an error if pos is out of bounds.

**NAME**

function::strlen — Returns the length of a string

**SYNOPSIS**

```plaintext
strlen:long(s:string)
```

**ARGUMENTS**

`s`

the string

**DESCRIPTION**

This function returns the length of the string, which can be zero up to MAXSTRINGLEN.

---

The function converts the string representation of a number to an integer. The `base` parameter indicates the number base to assume for the string (eg. 16 for hex, 8 for octal, 2 for binary).

**NAME**

function::strtol — strtol - Convert a string to a long

**SYNOPSIS**

```plaintext
strtol:long(str:string,base:long)
```

**ARGUMENTS**

`str`  
string to convert

`base`  
the base to use

**DESCRIPTION**

This function converts the string representation of a number to an integer. The `base` parameter indicates the number base to assume for the string (eg. 16 for hex, 8 for octal, 2 for binary).
NAME
function::substr — Returns a substring

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

ARGUMENTS
str
the string to take a substring from

start
starting position of the extracted string (first character is 0)

length
length of string to return

DESCRIPTION
Returns the substring of the given string at the given start position with the given length (or smaller if the
length of the original string is less than start + length, or length is bigger than MAXSTRINGLEN).

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

SYNOPSIS
text_str:string(input:string)

ARGUMENTS
input
the string to escape

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

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

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

ARGUMENTS

input
    the string to escape

len
    maximum length of string to return (0 implies MAXSTRINGLEN)

quoted
    put double quotes around the string. If input string is truncated it will have “...” after the second quote

DESCRIPTION

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

NAME

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

SYNOPSIS

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

ARGUMENTS

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

delim
    set of characters that delimit the tokens

DESCRIPTION

This function returns the next non-empty token in the given input string, where the tokens are delimited by characters in the delim string. If the input string is non-empty, it returns the first token. If the input string is empty, it returns the next token in the string passed in the previous call to tokenize. If no delimiter is found, the entire remaining input string is returned. It returns empty when no more tokens are available.
CHAPTER 30. UTILITY FUNCTIONS FOR USING ANSI CONTROL CHARS IN LOGS

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

NAME
function::ansi_clear_screen — Move cursor to top left and clear screen.

SYNOPSIS
ansi_clear_screen()

ARGUMENTS
None

DESCRIPTION
Sends ansi code for moving cursor to top left and then the ansi code for clearing the screen from the cursor position to the end.

NAME
function::ansi_cursor_hide — Hides the cursor.

SYNOPSIS
ansi_cursor_hide()

ARGUMENTS
None

DESCRIPTION
Sends ansi code for hiding the cursor.

NAME
function::ansi_cursor_move — Move cursor to new coordinates.

SYNOPSIS
ansi_cursor_move(x:long,y:long)

ARGUMENTS
x
   Row to move the cursor to.
y

Colomn to move the cursor to.

**DESCRIPTION**
Sends ansi code for positioning the cursor at row x and column y. Coordinates start at one, (1,1) is the top-left corner.

**NAME**
function::ansi_cursor_restore — Restores a previously saved cursor position.

**SYNOPSIS**

```c
ansi_cursor_restore()
```

**ARGUMENTS**
None

**DESCRIPTION**
Sends ansi code for restoring the current cursor position previously saved with `ansi_cursor_save`.

**NAME**
function::ansi_cursor_save — Saves the cursor position.

**SYNOPSIS**

```c
ansi_cursor_save()
```

**ARGUMENTS**
None

**DESCRIPTION**
Sends ansi code for saving the current cursor position.

**NAME**
function::ansi_cursor_show — Shows the cursor.

**SYNOPSIS**

```c
ansi_cursor_show()
```

**ARGUMENTS**
None
DESCRIPTION
Sends ansi code for showing the cursor.

NAME
function::ansi_new_line — Move cursor to new line.

SYNOPSIS
ansi_new_line()

ARGUMENTS
None

DESCRIPTION
Sends ansi code new line.

NAME
function::ansi_reset_color — Resets Select Graphic Rendition mode.

SYNOPSIS
ansi_reset_color()

ARGUMENTS
None

DESCRIPTION
Sends ansi code to reset foreground, background and color attribute to default values.

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

SYNOPSIS
ansi_set_color(fg:long)

ARGUMENTS
fg
  Foreground color to set.

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

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

SYNOPSIS

ansi_set_color2(fg:long, bg:long)

ARGUMENTS

fg
Foreground color to set.

bg
Background color to set.

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

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

SYNOPSIS

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

ARGUMENTS

fg
Foreground color to set.

bg
Background color to set.

attr
Color attribute to set.

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

NAME
function::indent — returns an amount of space to indent

SYNOPSIS
indent:string(delta:long)

ARGUMENTS

delta

the amount of space added/removed for each call

DESCRIPTION
This function returns a string with appropriate indentation. Call it with a small positive or matching
negative delta. Unlike the thread_indent function, the indent does not track individual indent values on a
per thread basis.

NAME
function::indent_depth — returns the global nested-depth

SYNOPSIS
indent_depth:long(delta:long)

ARGUMENTS

delta

the amount of depth added/removed for each call

DESCRIPTION
This function returns a number for appropriate indentation, similar to indent. Call it with a small positive
or matching negative delta. Unlike the thread_indent_depth function, the indent does not track individual
indent values on a per thread basis.

NAME
function::thread_indent — returns an amount of space with the current task information

SYNOPSIS
thread-indent: string(delta: long)

ARGUMENTS

delta
    the amount of space added/removed for each call

DESCRIPTION
This function returns a string with appropriate indentation for a thread. Call it with a small positive or
matching negative delta. If this is the real outermost, initial level of indentation, then the function resets
the relative timestamp base to zero. The timestamp is as per provided by the __indent_timestamp
function, which by default measures microseconds.

NAME
function::thread-indent-depth — returns the nested-depth of the current task

SYNOPSIS

thread-indent-depth: long(delta: long)

ARGUMENTS

delta
    the amount of depth added/removed for each call

DESCRIPTION
This function returns an integer equal to the nested function-call depth starting from the outermost initial
level. This function is useful for saving space (consumed by whitespace) in traces with long nested
function calls. Use this function in a similar fashion to thread-indent, i.e., in call-probe, use
thread-indent-depth(1) and in return-probe, use thread-indent-depth(-1)
CHAPTER 31. SYSTEMTAP TRANSLATOR TAPSET

This family of user-space probe points is used to probe the operation of the SystemTap translator (stap) and run command (staprun). The tapset includes probes to watch the various phases of SystemTap and SystemTap's management of instrumentation cache. It contains the following probe points:

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

SYNOPSIS
stap.cache_add_mod

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

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

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

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

SYNOPSIS
stap.cache_add_nss

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

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

DESCRIPTION
Fires just before the file is actually moved. Note: stap must compiled with NSS support; if moving the kernel module fails, this probe will not fire.
NAME
probe::stap.cache_add_src — Adding C code translation to cache

SYNOPSIS
   stap.cache_add_src

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

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

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

NAME
probe::stap.cache_clean — Removing file from stap cache

SYNOPSIS
   stap.cache_clean

VALUES
   path
       the path to the .ko/.c file being removed

DESCRIPTION
Fires just before the call to unlink the module/source file.

NAME
probe::stap.cache_get — Found item in stap cache

SYNOPSIS
   stap.cache_get

VALUES
   module_path
       the path of the .ko kernel module file
source_path

the path of the .c source file

DESCRIPTION
Fires just before the return of get_from_cache, when the cache grab is successful.

NAME
probe::stap.pass0 — Starting stap pass0 (parsing command line arguments)

SYNOPSIS
stap.pass0

VALUES

session

the systemtap_session variable s

DESCRIPTION
pass0 fires after command line arguments have been parsed.

NAME
probe::stap.pass0.end — Finished stap pass0 (parsing command line arguments)

SYNOPSIS
stap.pass0.end

VALUES

session

the systemtap_session variable s

DESCRIPTION
pass0.end fires just before the gettimeofday call for pass1.

NAME
probe::stap.pass1.end — Finished stap pass1 (parsing scripts)

SYNOPSIS
stap.pass1.end

VALUES

`session`
the systemtap_session variable s

DESCRIPTION

pass1.end fires just before the jump to cleanup if s.last_pass = 1.

NAME

probe::stap.pass1a — Starting stap pass1 (parsing user script)

SYNOPSIS

stap.pass1a

VALUES

`session`
the systemtap_session variable s

DESCRIPTION

pass1a fires just after the call to `gettimeofday`, before the user script is parsed.

NAME

probe::stap.pass1b — Starting stap pass1 (parsing library scripts)

SYNOPSIS

stap.pass1b

VALUES

`session`
the systemtap_session variable s

DESCRIPTION

pass1b fires just before the library scripts are parsed.
probe::stap.pass2 — Starting stap pass2 (elaboration)

SYNOPSIS

stap.pass2

VALUES

session

the systemtap_session variable s

DESCRIPTION

pass2 fires just after the call to gettimeofday, just before the call to semantic_pass.

NAME

probe::stap.pass2.end — Finished stap pass2 (elaboration)

SYNOPSIS

stap.pass2.end

VALUES

session

the systemtap_session variable s

DESCRIPTION

pass2.end fires just before the jump to cleanup if s.last_pass = 2

NAME

probe::stap.pass3 — Starting stap pass3 (translation to C)

SYNOPSIS

stap.pass3

VALUES

session

the systemtap_session variable s

DESCRIPTION

pass3 fires just after the call to gettimeofday, just before the call to translate_pass.
NAME
probe::stap.pass3.end — Finished stap pass3 (translation to C)

SYNOPSIS
stap.pass3.end

VALUES
session
the systemtap_session variable s

DESCRIPTION
pass3.end fires just before the jump to cleanup if s.last_pass = 3

NAME
probe::stap.pass4 — Starting stap pass4 (compile C code into kernel module)

SYNOPSIS
stap.pass4

VALUES
session
the systemtap_session variable s

DESCRIPTION
pass4 fires just after the call to gettimeofday, just before the call to compile_pass.

NAME
probe::stap.pass4.end — Finished stap pass4 (compile C code into kernel module)

SYNOPSIS
stap.pass4.end

VALUES
session
the systemtap_session variable s
DESCRIPTION
pass4.end fires just before the jump to cleanup if s.last_pass = 4

NAME
probe::stap.pass5 — Starting stap pass5 (running the instrumentation)

SYNOPSIS
stap.pass5

VALUES
session
the systemtap_session variable s

DESCRIPTION
pass5 fires just after the call to gettimeofday, just before the call to run_pass.

NAME
probe::stap.pass5.end — Finished stap pass5 (running the instrumentation)

SYNOPSIS
stap.pass5.end

VALUES
session
the systemtap_session variable s

DESCRIPTION
pass5.end fires just before the cleanup label

NAME
probe::stap.pass6 — Starting stap pass6 (cleanup)

SYNOPSIS
stap.pass6

VALUES
session
**DESCRIPTION**
pass6 fires just after the cleanup label, essentially the same spot as pass5.end

**NAME**
probe::stap.pass6.end — Finished stap pass6 (cleanup)

**SYNOPSIS**
stap.pass6.end

**VALUES**

<table>
<thead>
<tr>
<th>session</th>
</tr>
</thead>
<tbody>
<tr>
<td>the systemtap_session variable s</td>
</tr>
</tbody>
</table>

**DESCRIPTION**
pass6.end fires just before main's return.

**NAME**
probe::stap.system — Starting a command from stap

**SYNOPSIS**
stap.system

**VALUES**

<table>
<thead>
<tr>
<th>command</th>
</tr>
</thead>
<tbody>
<tr>
<td>the command string to be run by posix_spawn (as sh -c &lt;str&gt;)</td>
</tr>
</tbody>
</table>

**DESCRIPTION**
Fires at the entry of the stap_system command.

**NAME**
probe::stap.system.return — Finished a command from stap

**SYNOPSIS**
stap.system.return
VALUES

ret
a return code associated with running waitpid on the spawned process; a non-zero value indicates error

DESCRIPTION
Fires just before the return of the stap_system function, after waitpid.

NAME
probe::stap.system.spawn — stap spawned new process

SYNOPSIS

stap.system.spawn

VALUES

ret
the return value from posix_spawn

pid
the pid of the spawned process

DESCRIPTION
Fires just after the call to posix_spawn.

NAME
probe::stapio.receive_control_message — Received a control message

SYNOPSIS

stapio.receive_control_message

VALUES

len
the length (in bytes) of the data blob

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

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

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

NAME
probe::staprun.insert_module — Inserting SystemTap instrumentation module

SYNOPSIS
staprun.insert_module

VALUES
path
the full path to the .ko kernel module about to be inserted

DESCRIPTION
Fires just before the call to insert the module.

NAME
probe::staprun.remove_module — Removing SystemTap instrumentation module

SYNOPSIS
staprun.remove_module

VALUES
name
the stap module name to be removed (without the .ko extension)

DESCRIPTION
Fires just before the call to remove the module.

NAME
probe::staprun.send_control_message — Sending a control message

SYNOPSIS
staprun.send_control_message
VALUES

type

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

data

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

len

the length (in bytes) of the data blob

DESCRIPTION

Fires at the beginning of the send_request function.
CHAPTER 32. NETWORK FILE STORAGE TAPSETS

This family of probe points is used to probe network file storage functions and operations.

NAME
function::nfsderror — Convert nfsd error number into string

SYNOPSIS

ARGUMENTS

DESCRIPTION
This function returns a string for the error number passed into the function.

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

SYNOPSIS

VALUES

size
number of pages to be read in this execution

i_flag
file flags

file
file argument

ino
inode number

i_size
file length in bytes

dev
device identifier

\textit{rsize}

read size (in bytes)

\textit{__page}

the address of page

\textit{sb\_flag}

super block flags

\textit{page\_index}

offset within mapping, can used a page identifier and position identifier in the page frame

\textbf{DESCRIPTION}
Read the page over, only fires when a previous async read operation failed

\textbf{NAME}
probe::nfs.aop.readpages — NFS client reading multiple pages

\textbf{SYNOPSIS}
nfs.aop.readpages

\textbf{VALUES}

\textit{nr\_pages}

number of pages attempted to read in this execution

\textit{ino}

inode number

\textit{file}

filp argument

\textit{size}

number of pages attempted to read in this execution

\textit{rsize}

read size (in bytes)

\textit{dev}

device identifier
rpaces
read size (in pages)

DESCRIPTION
Fires when in readahead way, read several pages once

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

SYNOPSIS
nfs.aop.release_page

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

DESCRIPTION
Fires when do a release operation on NFS.

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

SYNOPSIS
nfs.aop.set_page_dirty

VALUES
__page
the address of page

`page_flag`
page flags

**DESCRIPTION**
This probe attaches to the generic `__set_page_dirty_nobuffers` function. Thus, this probe is going to fire on many other file systems in addition to the NFS client.

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

**SYNOPSIS**
```
nfs.aop.write_begin
```

**VALUES**

`__page`
the address of page

`page_index`
offset within mapping, can used a page identifier and position identifier in the page frame

`size`
write bytes

`to`
end address of this write operation

`ino`
inode number

`offset`
start address of this write operation

`dev`
device identifier

**DESCRIPTION**
Occurs when write operation occurs on nfs. It prepare a page for writing, look for a request corresponding to the page. If there is one, and it belongs to another file, it flush it out before it tries to copy anything into the page. Also do the same if it finds a request from an existing dropped page
NAME
probe:nfs.aop.write_end — NFS client complete writing data

SYNOPSIS
nfs.aop.write_end

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

DESCRIPTION
Fires when do a write operation on nfs, often after prepare_write
Update and possibly write a cached page of an NFS file.
NAME
probe::nfs.aop.writepage — NFS client writing a mapped page to the NFS server

SYNOPSIS
nfs.aop.writepage

VALUES

wsize
write size

size
number of pages to be written in this execution

i_flag
file flags

for_kupdate
a flag of writeback_control, indicates if it's a kupdate writeback

ino
inode number

i_size
file length in bytes

dev
device identifier

for_reclaim
a flag of writeback_control, indicates if it's invoked from the page allocator

__page
the address of page

sb_flag
super block flags

page_index
offset within mapping, can used a page identifier and position identifier in the page frame

i_state
inode state flags
DESCRIPTION
The priority of wb is decided by the flags `for_reclaim` and `for_kupdate`.

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

SYNOPSIS
nfs.aop.writepages

VALUES
`for_reclaim`
    a flag of writeback_control, indicates if it's invoked from the page allocator

`wpages`
    write size (in pages)

`nr_to_write`
    number of pages attempted to be written in this execution

`for_kupdate`
    a flag of writeback_control, indicates if it's a kupdate writeback

`ino`
    inode number

`size`
    number of pages attempted to be written in this execution

`wsize`
    write size

`dev`
    device identifier

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

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

SYNOPSIS
VALUES

<table>
<thead>
<tr>
<th>ino</th>
<th>inode number</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>cache_time</th>
<th>when we started read-caching this inode</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>file_name</th>
<th>file name</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>buf</th>
<th>the address of buf in user space</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>dev</th>
<th>device identifier</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>pos</th>
<th>current position of file</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>attrtimeo</th>
<th>how long the cached information is assumed to be valid. We need to revalidate the cached attrs for this inode if jiffies - read_cache_jiffies &gt; attrtimeo.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>count</th>
<th>read bytes</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>parent_name</th>
<th>parent dir name</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>cache_valid</th>
<th>cache related bit mask flag</th>
</tr>
</thead>
</table>

NAME

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

SYNOPSIS

| nfs.fop.aio_write |

VALUES

| count | 
**NAME**

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

**SYNOPSIS**

```
  nfs.fop.check_flags
```

**VALUES**

- `flag`
  - file flag

---

**NAME**

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

**SYNOPSIS**

```
  nfs.fop.flush
```

**VALUES**

- `ndirty`
number of dirty page

*ino*

inode number

*mode*

file mode

*dev*

device identifier

NAME

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

SYNOPSIS

nfs.fop.fsync

VALUES

*ndirty*

number of dirty pages

*ino*

inode number

*dev*

device identifier

NAME

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

SYNOPSIS

nfs.fop.llseek

VALUES

*ino*

inode number

*whence*
the position to seek from

devoffset
  device identifier
  the offset of the file will be repositioned
whelence_str
  symbolic string representation of the position to seek from

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

SYNOPSIS

| nfs.fop.lock |

VALUES

fl_start
  starting offset of locked region

ino
  inode number

fl_flag
  lock flags

i_mode
  file type and access rights

devoffset
  device identifier

fl_end
  ending offset of locked region

fl_type
  lock type

cmd
  cmd arguments
NAME
probe::nfs.fop.mmap — NFS client mmap operation

SYNOPSIS

    nfs.fop.mmap

VALUES

attrtimeo
    how long the cached information is assumed to be valid. We need to revalidate the cached attrs for
    this inode if jiffies - read_cache_jiffies > attrtimeo.

vm_end
    the first byte after end address within vm_mm

dev
    device identifier

buf
    the address of buf in user space

vm_flag
    vm flags

cache_time
    when we started read-caching this inode

file_name
    file name

ino
    inode number

cache_valid
    cache related bit mask flag

parent_name
    parent dir name

vm_start
    start address within vm_mm
NAME
probe::nfs.fop.open — NFS client file open operation

SYNOPSIS
nfs.fop.open

VALUES
flag
  file flag
i_size
  file length in bytes
dev
  device identifier
file_name
  file name
ino
  inode number

NAME
probe::nfs.fop.read — NFS client read operation

SYNOPSIS
nfs.fop.read

VALUES
devname
  block device name

DESCRIPTION
SystemTap uses the vfs.do_sync_read probe to implement this probe and as a result will get operations other than the NFS client read operations.

NAME
probe::nfs.fop.read_iter — NFS client read_iter file operation
**SYNOPSIS**

nfs.fop.read_iter

**VALUES**

*ino*

inode number

*file_name*

file name

*cache_time*

when we started read-caching this inode

*pos*

current position of file

*dev*

device identifier

*attrtimeo*

how long the cached information is assumed to be valid. We need to revalidate the cached attrs for this inode if jiffies - read_cache_jiffies > attrtimeo.

*count*

read bytes

*parent_name*

parent dir name

*cache_valid*

cache related bit mask flag

---

**NAME**

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

**SYNOPSIS**

nfs.fop.release

**VALUES**

*ino*
in node number

dev
    device identifier

mode
    file mode

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

SYNOPSIS

| nfs.fop.sendfile |

VALUES

cache_valid
    cache related bit mask flag

ppos
    current position of file

count
    read bytes

dev
    device identifier

attrtimeo
    how long the cached information is assumed to be valid. We need to revalidate the cached attrs for this inode if jiffies - read_cache_jiffies > attrtimeo.

ino
    inode number

cache_time
    when we started read-caching this inode

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

nfs.fop.write

VALUES

devname
    block device name

DESCRIPTION
SystemTap uses the vfs.do_sync_write probe to implement this probe and as a result will get operations other than the NFS client write operations.

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

SYNOPSIS

nfs.fop.write_iter

VALUES

parent_name
    parent dir name

count
    read bytes

pos
    offset of the file

dev
    device identifier

file_name
    file name

ino
    inode number

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

nfs.proc.commit

VALUES

size
  read bytes in this execution

prot
  transfer protocol

version
  NFS version

server_ip
  IP address of server

bitmask1
  V4 bitmask representing the set of attributes supported on this filesystem

offset
  the file offset

bitmask0
  V4 bitmask representing the set of attributes supported on this filesystem

DESCRIPTION

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

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

NAME

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

SYNOPSIS

nfs.proc.commit_done

VALUES

status
  result of last operation
server_ip
   IP address of server

prot
   transfer protocol

version
   NFS version

count
   number of bytes committed

valid
   fattr->valid, indicates which fields are valid

timestamp
   V4 timestamp, which is used for lease renewal

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

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

SYNOPSIS
   nfs.proc.commit_setup

VALUES

version
   NFS version

count
   bytes in this commit

prot
   transfer protocol

server_ip
   IP address of server

bitmask1
V4 bitmask representing the set of attributes supported on this filesystem

**bitmask0**
V4 bitmask representing the set of attributes supported on this filesystem

**offset**
the file offset

**size**
bytes in this commit

---

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

---

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

**SYNOPSIS**
```
nfs.proc.create
```

**VALUES**

**server_ip**
IP address of server

**prot**
transfer protocol

**version**
NFS version (the function is used for all NFS version)

**filename**
file name

**fh**
file handle of parent dir

**filelen**
length of file name

**flag**
indicates create mode (only for NFSv3 and NFSv4)
NAME
probe::nfs.proc.handle_exception — NFS client handling an NFSv4 exception

SYNOPSIS
nfs.proc.handle_exception

VALUES
errorcode
  indicates the type of error

DESCRIPTION
This is the error handling routine for processes for NFSv4.

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

SYNOPSIS
nfs.proc.lookup

VALUES
bitmask1
  V4 bitmask representing the set of attributes supported on this filesystem

bitmask0
  V4 bitmask representing the set of attributes supported on this filesystem

filename
  the name of file which client opens/searches on server

server_ip
  IP address of server

prot
  transfer protocol

name_len
  the length of file name

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

SYNOPSIS
nfs.proc.open

VALUES
flag
file flag

filename
file name

version
NFS version (the function is used for all NFS version)

prot
transfer protocol

mode
file mode

server_ip
IP address of server

DESCRIPTION
Allocate file read/write context information

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

SYNOPSIS
nfs.proc.read

VALUES
offset
the file offset
**server_ip**

IP address of server

**flags**

used to set task->tk_flags in rpc_init_task function

**prot**

transfer protocol

**count**

read bytes in this execution

**version**

NFS version

**DESCRIPTION**

All the nfs.proc.read kernel functions were removed in kernel commit 8e0969 in December 2006, so these probes do not exist on Linux 2.6.21 and newer kernels.

**NAME**

probe::nfs.proc.read_done — NFS client response to a read RPC task

**SYNOPSIS**

```
nfs.proc.read_done
```

**VALUES**

**timestamp**

V4 timestamp, which is used for lease renewal

**prot**

transfer protocol

**count**

number of bytes read

**version**

NFS version

**status**

result of last operation

**server_ip**
IP address of server

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

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

SYNOPSIS
nfs.proc.read_setup

VALUES

* offset
  the file offset

* server_ip
  IP address of server

* prot
  transfer protocol

* version
  NFS version

* count
  read bytes in this execution

* size
  read bytes in this execution

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

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

SYNOPSIS
nfs.proc.release
VALUES

flag
  file flag

filename
  file name

prot
  transfer protocol

version
  NFS version (the function is used for all NFS version)

mode
  file mode

server_ip
  IP address of server

DESCRIPTION
Release file read/write context information

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

SYNOPSIS

nfs.proc.remove

VALUES

prot
  transfer protocol

version
  NFS version (the function is used for all NFS version)

server_ip
  IP address of server

filelen
  length of file name
filename
  file name

fh
  file handle of parent dir

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

SYNOPSIS
  nfs.proc.rename

VALUES
  new_fh
    file handle of new parent dir
  new_filelen
    length of new file name
  old_name
    old file name
  version
    NFS version (the function is used for all NFS version)
  old_fh
    file handle of old parent dir
  prot
    transfer protocol
  new_name
    new file name
  old_filelen
    length of old file name
  server_ip
    IP address of server
NAME
probe::nfs.proc.rename_done — NFS client response to a rename RPC task

SYNOPSIS

VALUES

\texttt{timestamp}

V4 timestamp, which is used for lease renewal

\texttt{status}

result of last operation

\texttt{server_ip}

IP address of server

\texttt{prot}

transfer protocol

\texttt{version}

NFS version

\texttt{old_fh}

file handle of old parent dir

\texttt{new_fh}

file handle of new parent dir

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

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

SYNOPSIS

VALUES

\texttt{fh}

file handle of parent dir
The rename_setup function is used to setup a rename RPC task. It is not doing the actual rename operation.

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

**SYNOPSIS**
```nfs.proc.write```

**VALUES**
- **size**
  - read bytes in this execution
- **flags**
  - used to set task->tk_flags in rpc_init_task function
- **prot**
  - transfer protocol
- **version**
  - NFS version
- **bitmask1**
  - V4 bitmask representing the set of attributes supported on this filesystem
- **offset**
  - the file offset
- **bitmask0**
  - V4 bitmask representing the set of attributes supported on this filesystem
- **server_ip**
  - IP address of server
DESCRIPTION
All the nfs.proc.write kernel functions were removed in kernel commit 200baa in December 2006, so these probes do not exist on Linux 2.6.21 and newer kernels.

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

SYNOPSIS
nfs.proc.write_done

VALUES
server_ip
  IP address of server

status
  result of last operation

version
  NFS version

count
  number of bytes written

prot
  transfer protocol

valid
  fatr->valid, indicates which fields are valid

timestamp
  V4 timestamp, which is used for lease renewal

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

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

nfs.proc.write_setup

VALUES

size
bytes written in this execution

prot
transfer protocol

version
NFS version

count
bytes written in this execution

bitmask0
V4 bitmask representing the set of attributes supported on this filesystem

bitmask1
V4 bitmask representing the set of attributes supported on this filesystem

offset
the file offset

how
used to set args.stable. The stable value could be: NFS_UNSTABLE,NFS_DATA_SYNC,NFS_FILE_SYNC (in nfs.proc3.write_setup and nfs.proc4.write_setup)

server_ip
IP address of server

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

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

SYNOPSIS

nfsd.close
VALUES

filename
  file name

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

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

SYNOPSIS
nfsd.commit

VALUES

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

flag
  indicates whether this execution is a sync operation

offset
  the offset of file

size
  read bytes

count
  read bytes

client_ip
  the ip address of client

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

SYNOPSIS
nfsd.create
VALUES

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

iap_valid
Attribute flags

filelen
the length of file name

type
file type(regular,dir,device,fifo ...)

filename
file name

iap_mode
file access mode

client_ip
the ip address of client

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

NAME
probe::nfsd.createv3 — NFS server creating a regular file or set file attributes for client

SYNOPSIS

nfsd.createv3

VALUES

iap_mode
file access mode

filename
file name

client_ip
the ip address of client
fh
  file handle (the first part is the length of the file handle)

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

filelen
  the length of file name

iap_valid
  Attribute flags

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

truncp
  trunp arguments, indicates if the file should be truncated

DESCRIPTION
This probe points is only called by nfsd3_proc_create and nfsd4_open when op_claim_type is
NFS4_OPEN_CLAIM_NULL.

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

SYNOPSIS
nfsd.dispatch

VALUES
xid
  transmission id

version
  nfs version

proto
  transfer protocol

proc
  procedure number

client_ip
the ip address of client

prog
program number

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

SYNOPSIS

nfsd.lookup

VALUES

filename
file name

client_ip
the ip address of client

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

filelen
the length of file name

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

SYNOPSIS

nfsd.open

VALUES

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

type
type of file (regular file or dir)

access
indicates the type of open (read/write/commit/readdir...)

client_ip  
the ip address of client

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

SYNOPSIS

| nfsd.proc.commit |

VALUES  

count  
read bytes

client_ip  
the ip address of client

proto  
transfer protocol

size  
read bytes

version  
nfs version

uid  
requester's user id

offset  
the offset of file

gid  
requester's group id

fh  
file handle (the first part is the length of the file handle)
NAME
probe::nfsd.proc.create — NFS server creating a file for client

SYNOPSIS
nfsd.proc.create

VALUES
proto
    transfer protocol

filename
    file name

client_ip
    the ip address of client

uid
    requester's user id

version
    nfs version

gid
    requester's group id

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

filelen
    length of file name

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

SYNOPSIS
nfsd.proc.lookup

VALUES
fh
    file handle of parent dir (the first part is the length of the file handle)
**NAME**
probe::nfsd.proc.read — NFS server reading file for client

**SYNOPSIS**
nfsd.proc.read

**VALUES**

`size`
read bytes

`vec`
struct kvec, includes buf address in kernel address and length of each buffer

`version`
nfs version

`uid`
requester's user id

`count`
read bytes
client_ip
   the ip address of client

proto
   transfer protocol

offset
   the offset of file

gid
   requester's group id

vlen
   read blocks

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

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

SYNOPSIS
nfsd.proc.remove

VALUES

gid
   requester's group id

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

filelen
   length of file name

uid
   requester's user id

version
   nfs version

proto
   transfer protocol
filename
define filename

client_ip
define client_ip

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

SYNOPSIS
nfsd.proc.rename

VALUES
uid
define uid
requester’s user id
tfh
define tfh
file handler of new path
tname
define tname
new file name
filename
define filename
old file name
client_ip
define client_ip
the ip address of client
flen
define flen
length of old file name
gid
define gid
requester’s group id
fh
define fh
file handler of old path
tlen
define tlen
length of new file name
NAME
probe::nfsd.proc.write — NFS server writing data to file for client

SYNOPSIS

nfsd.proc.write

VALUES

offset
  the offset of file

gid
  requester's group id

vlen
  read blocks

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

size
  read bytes

vec
  struct kvec, includes buf address in kernel address and length of each buffer

stable
  argp->stable

version
  nfs version

uid
  requester's user id

count
  read bytes

client_ip
  the ip address of client

proto
  transfer protocol
NAME
probe::nfsd.read — NFS server reading data from a file for client

SYNOPSIS

nfsd.read

VALUES

offset
  the offset of file

vlen
  read blocks

file
  argument file, indicates if the file has been opened.

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

count
  read bytes

client_ip
  the ip address of client

size
  read bytes

vec
  struct kvec, includes buf address in kernel address and length of each buffer

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

SYNOPSIS

nfsd.rename

VALUES

tlen
  length of new file name
fh
  file handler of old path

flen
  length of old file name

client_ip
  the ip address of client

filename
  old file name

tname
  new file name

tfh
  file handler of new path

---

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

SYNOPSIS

nfsd.unlink

VALUES

filelen
  the length of file name

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

type
  file type (file or dir)

client_ip
  the ip address of client

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

SYNOPSIS

nfsd.write

VALUES

offset
  the offset of file

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

client_ip
  the ip address of client

vlen
  read blocks

file
  argument file, indicates if the file has been opened.

count
  read bytes

size
  read bytes

vec
  struct kvec, includes buf address in kernel address and length of each buffer
CHAPTER 33. SPECULATION

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

NAME
function::commit — Write out all output related to a speculation buffer

SYNOPSIS
commit(id:long)

ARGUMENTS
id
of the buffer to store the information in

DESCRIPTION
Output all the output for id in the order that it was entered into the speculative buffer by speculative.

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

SYNOPSIS
discard(id:long)

ARGUMENTS
id
of the buffer to store the information in

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

SYNOPSIS
speculate(id:long, output:string)

ARGUMENTS
id
buffer id to store the information in
**output**

string to write out when commit occurs

---

**DESCRIPTION**

Add a string to the speculative buffer for id.

---

**NAME**

function::speculation — Allocate a new id for speculative output

---

**SYNOPSIS**

```speculation:long()```

---

**ARGUMENTS**

None

---

**DESCRIPTION**

The `speculation` function is called when a new speculation buffer is needed. It returns an id for the speculative output. There can be multiple threads being speculated on concurrently. This id is used by other speculation functions to keep the threads separate.
CHAPTER 34. JSON TAPSET

This family of probe points, functions, and macros is used to output data in JSON format. It contains the following probe points, functions, and macros:

NAME
function::json_add_array — Add an array

SYNOPSIS

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

ARGUMENTS

- **name**
  - The name of the array.

- **description**
  - Array description. An empty string can be used.

DESCRIPTION

This function adds a array, setting up everything needed. Arrays contain other metrics, added with `json_add_array_numeric_metric` or `json_add_array_string_metric`.

NAME
function::json_add_array_numeric_metric — Add a numeric metric to an array

SYNOPSIS

`json_add_array_numeric_metric:long(array_name:string,metric_name:string,metric_description:string,metric_units:string)`

ARGUMENTS

- **array_name**
  - The name of the array the numeric metric should be added to.

- **metric_name**
  - The name of the numeric metric.

- **metric_description**
  - Metric description. An empty string can be used.

- **metric_units**
  - Metric units. An empty string can be used.
DESCRIPTION
This function adds a numeric metric to an array, setting up everything needed.

NAME
function::json_add_array_string_metric — Add a string metric to an array

SYNOPSIS

```
json_add_array_string_metric:long(array_name:string,metric_name:string,metric_description:string)
```

ARGUMENTS

array_name
The name of the array the string metric should be added to.

metric_name
The name of the string metric.

metric_description
Metric description. An empty string can be used.

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

NAME
function::json_add_numeric_metric — Add a numeric metric

SYNOPSIS

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

ARGUMENTS

name
The name of the numeric metric.

description
Metric description. An empty string can be used.

units
Metric units. An empty string can be used.
DESCRIPTION
This function adds a numeric metric, setting up everything needed.

NAME
function::json_add_string_metric — Add a string metric

SYNOPSIS

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

ARGUMENTS

name
   The name of the string metric.

description
   Metric description. An empty string can be used.

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

NAME
function::json_set_prefix — Set the metric prefix.

SYNOPSIS

json_set_prefix:long(prefix:string)

ARGUMENTS

prefix
   The prefix name to be used.

DESCRIPTION
This function sets the “prefix”, which is the name of the base of the metric hierarchy. Calling this function is optional, by default the name of the systemtap module is used.

NAME
macro::json_output_array_numeric_value — Output a numeric value for metric in an array.

SYNOPSIS
@json_output_array_numeric_value(array_name, array_index, metric_name, value)

**ARGUMENTS**

- **array_name**
  - The name of the array.

- **array_index**
  - The array index (as a string) indicating where to store the numeric value.

- **metric_name**
  - The name of the numeric metric.

- **value**
  - The numeric value to output.

**DESCRIPTION**

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

---

**NAME**

`macro::json_output_array_string_value` — Output a string value for metric in an array.

**SYNOPSIS**

@json_output_array_string_value(array_name, array_index, metric_name, value)

**ARGUMENTS**

- **array_name**
  - The name of the array.

- **array_index**
  - The array index (as a string) indicating where to store the string value.

- **metric_name**
  - The name of the string metric.

- **value**
  - The string value to output.

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

NAME
macro::json_output_data_end — End the json output.

SYNOPSIS
@json_output_data_end()

ARGUMENTS
None

DESCRIPTION
The json_output_data_end macro is designed to be called from the 'json_data' probe from the user's script. It marks the end of the JSON output.

NAME
macro::json_output_data_start — Start the json output.

SYNOPSIS
@json_output_data_start()

ARGUMENTS
None

DESCRIPTION
The json_output_data_start macro is designed to be called from the 'json_data' probe from the user's script. It marks the start of the JSON output.

NAME
macro::json_output_numeric_value — Output a numeric value.

SYNOPSIS
@json_output_numeric_value(name, value)

ARGUMENTS

name
The name of the numeric metric.
**value**

The numeric value to output.

**DESCRIPTION**
The `json_output_numeric_value` macro is designed to be called from the `json_data` probe in the user's script to output a metric's numeric value. This metric should have been added with `json_add_numeric_metric`.

**NAME**

`macro::json_output_numeric_value` — Output a numeric value.

**SYNOPSIS**

```plaintext
@json_output_numeric_value(name, value)
```

**ARGUMENTS**

- `name`
  - The name of the numeric metric.

- `value`
  - The numeric value to output.

**DESCRIPTION**
The `json_output_numeric_value` macro is designed to be called from the `json_data` probe in the user's script to output a metric's numeric value. This metric should have been added with `json_add_numeric_metric`.

**NAME**

`probe::json_data` — Fires whenever JSON data is wanted by a reader.

**SYNOPSIS**

```plaintext
json_data
```

**VALUES**

None

**CONTEXT**

This probe fires when the JSON data is about to be read. This probe must gather up data and then call the following macros to output the data in JSON format. First, `@json_output_data_start` must be called. That call is followed by one or more of the following (one call for each data item):
@json_output_string_value, @json_output_numeric_value, 
@json_output_array_string_value, and @json_output_array_numeric_value. Finally 
@json_output_data_end must be called.
CHAPTER 35. OUTPUT FILE SWITCHING TAPSET

Utility function to allow switching of output files.

NAME
function::switch_file — switch to the next output file

SYNOPSIS

switch_file()

ARGUMENTS
None

DESCRIPTION
This function sends a signal to the stapio process, commanding it to rotate to the next output file when output is sent to file(s).
## APPENDIX A. REVISION HISTORY

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Author</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-6</td>
<td>Tue Oct 30 2018</td>
<td>Vladimír Slávik</td>
<td>Release for Red Hat Enterprise Linux 7.6 GA.</td>
</tr>
<tr>
<td>7-5</td>
<td>Tue Jan 09 2018</td>
<td>Vladimír Slávik</td>
<td>Release for Red Hat Enterprise Linux 7.5 Beta.</td>
</tr>
</tbody>
</table>