bfdmux

Barrelfish Demultiplexer

Amin Baumeler, Rainer Voigt

16 Sep 2009

Contents

1 Barrelfish demultiplexer - bfdmux **1**

1.1 Introduction 1

1.2 Documentation 1

2 Bfdmux Filter Language **3**

2.1 Operators 4

2.2 Arithmetic 4

2.3 Comparison 4

2.4 Logical 4

2.5 Bitwise 5

2.6 Packet access 5

2.7 Examples 5

2.8 Operator precedence 5

3 Bfdmux development manual **7**

3.1 Communication concept 8

3.2 Important data fields 8

3.3 Functional overview 9

3.4 Event overview 10

4 Bfdmux network interface manual **11**

4.1 a new network interface 12

4.2 incoming data to bfdmux 12

5 Libbfdmux application developer manual **15**

5.1 Building libbfdmux 16

5.2 Binding your application together with libbfdmux 16

5.3 Interface to bfdmux 16
5.4 Managing debug output 17

6 Storage of filter code **19**
6.1 General 20
6.2 Op-Codes 20

7 Libbfdmux design **23**
7.1 Client interface protocol 24
7.2 Command packets 24
7.3 Command channel 25
7.4 Data channel 25
7.5 Read-write locks 26
7.6 Helper functions 26

8 Libbfdmux internal development manual **27**
8.1 Adding a command to libbfdmux 28

9 Sample libbfdmux applications **29**
9.1 Overview 30
9.2 Bfdmuxchat 30
9.3 Bfdmuxsniff 30
9.4 Message Queue Loopback 30

10 Todo List **33**

11 Directory Hierarchy **35**
11.1 Directories 35

12 Data Structure Index **37**
12.1 Data Structures 37

13 File Index **39**
13.1 File List 39

14 Directory Documentation **41**
14.1 bfdmux/ Directory Reference 41
14.2 libbfdmux/bfdmuxchat/ Directory Reference 42

14.3 libbfdmux/bfdmuxsniff/bfdmuxinject/ Directory Reference 43
14.4 libbfdmux/bfdmuxsniff/ Directory Reference 44
14.5 doc/ Directory Reference 45
14.6 libbfdmux/src/include/ Directory Reference 46
14.7 bfdmux/src/include/ Directory Reference 47
14.8 libbfdmux/ Directory Reference 48
14.9 libbfdmux/bfdmuxchat/msgq_loopback/ Directory Reference 49
14.10 bfdmux/src/netif/ Directory Reference 50
14.11 bfdmux/src/include/netif/ Directory Reference 51
14.12 libbfdmux/src/ Directory Reference 52
14.13 libbfdmux/bfdmuxsniff/src/ Directory Reference 53
14.14 libbfdmux/bfdmuxsniff/bfdmuxinject/src/ Directory Reference 54
14.15 libbfdmux/bfdmuxchat/msgq_loopback/src/ Directory Reference . . . 55
14.16 bfdmux/src/ Directory Reference 56

15 Data Structure Documentation **57**

15.1 client_app Struct Reference 57
15.2 cmd_attach Struct Reference 59
15.3 cmd_attach_answer Struct Reference 60
15.4 cmd_detach Struct Reference 61
15.5 cmd_detach_answer Struct Reference 62
15.6 cmd_error Struct Reference 63
15.7 cmd_recv Struct Reference 64
15.8 cmd_recv_answer Struct Reference 65
15.9 cmd_register Struct Reference 66
15.10 cmd_register_answer Struct Reference 67
15.11 cmd_send Struct Reference 68
15.12 cmd_send_answer Struct Reference 69
15.13 cmd_unregister Struct Reference 70
15.14 cmd_unregister_answer Struct Reference 71
15.15 filter Struct Reference 72
15.16 nic_message_buf Struct Reference 73
15.17 op_def_t Struct Reference 74

16 File Documentation **75**

16.1 bfdmux/src/bfdmux.c File Reference 75
16.2 bfdmux/src/codegen.c File Reference 82
16.3 bfdmux/src/filter.c File Reference 87
16.4 bfdmux/src/include/codegen.h File Reference 90
16.5 bfdmux/src/include/filter.h File Reference 93
16.6 bfdmux/src/include/netif.h File Reference 98
16.7 bfdmux/src/include/netif/mqif.h File Reference 101
16.8 bfdmux/src/include/opdefs.h File Reference 102
16.9 bfdmux/src/include/register.h File Reference 104
16.10 bfdmux/src/include/server.h File Reference 109
16.11 bfdmux/src/include/vm.h File Reference 113
16.12 bfdmux/src/netif/mqif.c File Reference 115
16.13 bfdmux/src/opdefs.c File Reference 119
16.14 bfdmux/src/register.c File Reference 122
16.15 bfdmux/src/server.c File Reference 126
16.16 bfdmux/src/vm.c File Reference 134
16.17 libbfdmux/bfdmuxchat/bfdmuxchat.c File Reference 137
16.18 libbfdmux/bfdmuxchat/msgq_loopback/src/msgq_clear.c File Reference 140
16.19 libbfdmux/bfdmuxchat/msgq_loopback/src/msgq_loopback.c File Reference 141
16.20 libbfdmux/bfdmuxsniff/bfdmuxinject/src/bfdmuxinject.c File Reference 142
16.21 libbfdmux/bfdmuxsniff/src/bfdmuxsniff.c File Reference 145
16.22 libbfdmux/src/bfdmux_ciprot.c File Reference 149
16.23 libbfdmux/src/include/bfdmux.h File Reference 152
16.24 libbfdmux/src/include/bfdmux_ciprot.h File Reference 155
16.25 libbfdmux/src/include/debug.h File Reference 161
16.26 libbfdmux/src/include/libbfdmux.h File Reference 163
16.27 libbfdmux/src/include/rwlock.h File Reference 168
16.28 libbfdmux/src/include/tools.h File Reference 173
16.29 libbfdmux/src/libbfdmux.c File Reference 178
16.30 libbfdmux/src/rwlock.c File Reference 187
16.31 libbfdmux/src/tools.c File Reference 192

Chapter 1

Barrelfish demultiplexer - bfdmux

1.1 Introduction

This is the documentation for the bfdmux project started by Amin Baumeler and Rainer Voigt in 2009 at ETH Zurich. bfdmux itself is a flexible network packet demultiplexer. The project also provides a library (libbfdmux) that simplifies the task of connecting applications to bfdmux.

1.2 Documentation

1.2.1 Client side

- How to develop applications that connect to bfdmux?
  See Libbfdmux application developer manual.
- How can packet filters be specified?
  See Bfdmux Filter Language.
- How does compiled filter code look like?
  See Storage of filter code.
- How to modify and extend libbfdmux itself?
  See Libbfdmux design
  and Libbfdmux internal development manual.
- Sample applications
  See Sample libbfdmux applications

1.2.2 Server side

- How can I connect a NIC to bfdmux?
  See Bfdmux network interface manual.

- How to modify and extend bfdmux itself?
  See Bfdmux development manual.

Chapter 2

Bfdmux Filter Language

2.1 Operators

Note:

Documentation on compiled code can be found at Storage of filter code.

2.2 Arithmetic

- + Addition
- - Subtraction
- ∗ Multiplication
- / Integer division
- % Modulus

2.3 Comparison

- == Equal
- > Signed greater
- < Signed less
- } Unsigned greater
- { Unsigned less
- >= Signed greater or equal
- <= Signed less or equal
- }= Unsigned greater or equal
- {= Unsigned less or equal
- != Unequal

2.4 Logical

- ! Not
- && And
- || Or

2.5 Bitwise

- & And
- | Or
- ∼ Not
- ^ Xor

2.6 Packet access

Storage types

- int8
- int16
- int32
- int64

2.7 Examples

2.7.1 Byte Access

- Access the first 16 bits of a packet: `int16[0]`
- Access a 32 bit value starting at byte 8 in the packet: `int32[8]`

2.7.2 Example

- `((int8[0] + int8[1]) == 5) && (int16[0] >= 32))`
  matches packets starting with 0x0104, 0x0203, 0x0302, 0x0401 and 0x0500. It will not match 0x06FF because all calculations are done as 64bit values!

Note:

It is actually possible to use round and square brackets interchangably, but we recommend using square brackets around packet addresses for clarity.

2.8 Operator precedence

Sorted from lowest to highest:

||, &&, |, ^, &, ==, !=, >=, <=, }=, {=, >, <, }, {, +, -, ∗, /, %, !, ∼, int8, int16, int32, int64

Chapter 3

Bfdmux development manual

3.1 Communication concept

Bfdmux communicates with it's client applications via the command interface, a simple unix socket. The data transmission is done using two shared memory segments, one for incoming and one for outgoing data. The shared memory holds at most one valid packet at a time. Whenever a segment is filled with new data, this is signalled to the other party via a special command. When the data has been read and processed, another command is issued to indicate that the buffer is free again. The following picture illustrates the relation of applications and bfdmux.

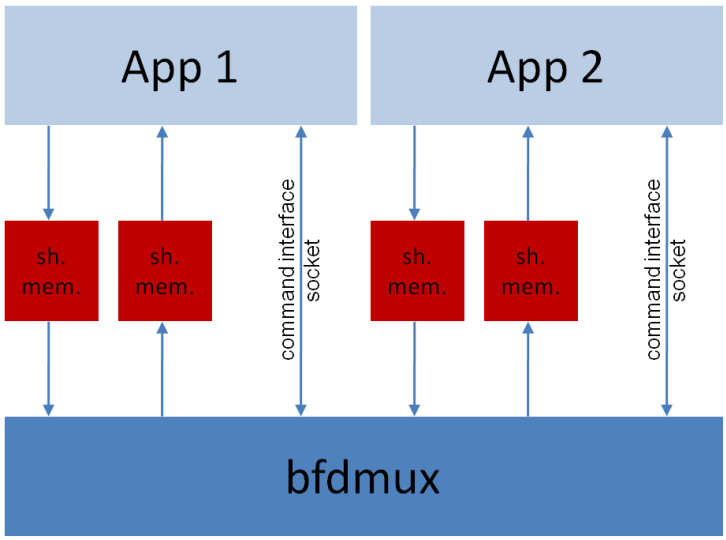


Figure 3.1: Application interface

3.2 Important data fields

Most important during bfdmux's operation is the field app_table. It contains all runtime information on the connected (and maybe registered) client applications. Please see the doxygen documentation for app_table to know more about it's members. The pointer structure of app_table and it's substructures is also visualized separately in the following image.

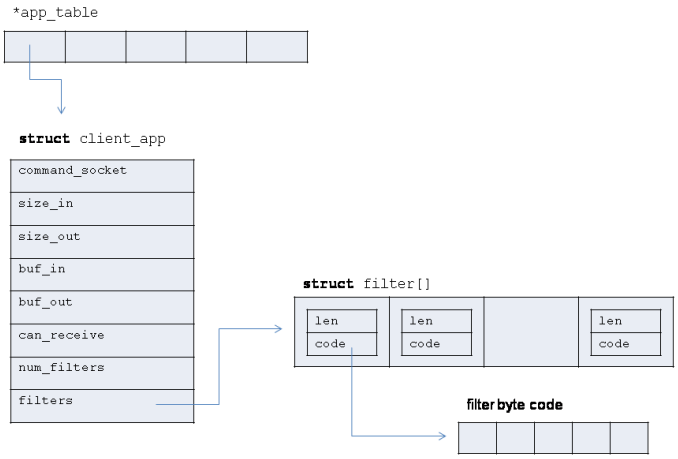


Figure 3.2: Application table layout

The image is in png format and thus cannot be displayed here. Please consult the html version of the documentation or look for 'app_table.png' in the subversion repository.

Note:

app_table contains num_apps entries. Every access needs to be locked by acquiring app_table_lock. See rwlock.c for details on locking.

Another field that is important at runtime is queue. This is the ring buffer used to store pending demultiplexing requests. queue_first contains the index of the first element in the queue, queue_len indicates the number of valid entries starting from queue_first.

Note:

If queue_len is 0, queue_first points to the next empty position. Otherwise queue_first contains the index of the first valid item (which is the one to be processed next). All accesses to queue need to be locked using rwlock.c with the lock identifier queue_lock.

3.3 Functional overview

Bfdmux itself uses two threads at runtime. First, the main thread spawns the server thread that handles all client communication over the unix sockets. Then, then main thread enters a polling loop and waits for new demultiplexing requests. Whenever a request is queued by the NIC, it runs the filter on the packet data and copies the packet into the receiving applications buffers.

The server thread listens for connecting clients and waits for commands on all existing client sockets using the select() function.

3.4 Event overview

filter.c:new_packet_event() invokes all filters on the given packet and forwards it to the corresponding applications. This method is only called by the bfdmux main thread.

server.c:new_message_event() invoked by the server thread whenever a new command has arrived from one of the client applications.

server.c:new_client_event() invoked by the server thread whenever a new application tries to establish a command connection to bfdmux's socket.

Chapter 4

Bfdmux network interface manual

4.1 a new network interface

To use bfdmux with a specific NIC one has to implement some basic interface functions as declared in bfdmux/src/include/netif.h. As netif.h declares the necessary functions it is best to define them in a separate c file that will later be linked into the bfdmux executable.

err_t init_nic_interface();

This method is called during bfdmux initialization. This is the time to talk to your NIC and prepare it for it's job. You might install a signal or interrupt handler for incoming data at this point. There is no special interaction with bfdmux happening here.

err_t close_nic_interface();

This procedure is invoked during bfdmux shutdown. Cleanup everything that has been set up in init_nic_interface() before and remove the interrupt handler, if any. There is no interaction with bfdmux.

err_t nic_send(void* buf, size_t len);

This function takes a buffer of specified size and transmits it over the network. The function will be called directly after the client signalled outgoing data to bfdmux. Therefore the call should return immediately to not block bfdmux's operation.

4.2 incoming data to bfdmux

Usually the function init_nic_interface will install some kind of signal or interrupt handler to be called if new data has arrived on the network interface. The example interface in mqif.c sets up a unix signal handler and reads data from a message queue whenever it fires (see signal_handler in mqif.c). If not already done by the NIC, the handler should copy the data into the ram. Then it just calls the function demux() declared in bfdmux.h. demux() returns true, if the packet could be added to the demultiplexing queue and false, if the queue is already full. In the latter case, the handler can either queue the packet by himself or just drop it. Please keep in mind that demux() returns immediately. If it returns true, the packet is waiting to be processed and cannot be moved or deleted in ram (see warning)!

The demultiplexing queue is defined in bfdmux.h. It is a ring buffer storing packet addresses and lengths for all pending demux requests. Therefore no data is copied when queueing a request. The size of the ring buffer is PROC_QUEUE_LEN and can be set to one, to entirely disable request queueing in bfdmux.

Note:

We found the request queueing to be useful to handle bursts. As bfdmux should provide a demux function to the interrupt handler that returns non-blocking, the actual packet demultiplexing happens in a separate thread (the bfdmux main thread). Thus, whenever two or more packets arrived at the same time, the demultiplexer could only proocess the first packet, if there was no queueing. Here 'same time' means 'too short that the demultiplexing thread could be sheduled inbetween'. This leads to a high number of dropped packets, even though the actual packet

demultiplexing was, on average, fast enough to process all incoming packets.

Warning:

In the current version of bfdmux there is no feedback to the nic interface when the packet was finally processed by bfdmux (feedback could be implemented at the end of new_packet_event). As the requests reside in a ring buffer of size PROC_-QUEUE_LEN, the nic interrupt handler should ensure that at least the last PROC_-QUEUE_LEN packets that have been sucessfully enqueued using demux() stay at their originally specified address in ram and will not be overridden during demultiplexing. If packets are overwritten during demultiplexing, bfdmux might forward non-matching packets to it's applications, because the filter has been run on the original packet and the packet has been overwritten before bfdmux finally copies it to the applications memory.

Chapter 5

Libbfdmux application developer manual

5.1 Building libbfdmux

To build libbfdmux enter the libbfdmux/src directory and execute

$ make depend

followed by

$ make

To clean the already built objects you can run

$ make clean

5.2 Binding your application together with libbfdmux

To include this library in your project you need to include the "libbfdmux.h" header file and to add the libbfdmux/src/include directory to your include path as a pre-processor option. Additionally you will have to link your objects against bfdmux_ciprot.o and libbfdmux.o and you also need the pthread library for successful compilation.

5.3 Interface to bfdmux

libbfdmux enables your application to communicate with the bfdmux instance running on your machine to receive and send network packets.

5.3.1 Application registration

First the applications needs to register itself with the bfdmux process using the 'register_app()' function. This function takes a pointer to a callback function that will be called to handle incoming network packets. As additional arguments, it expects two uninitailized pointers to an inbound and outbound buffer pointer, and the required buffer sizes. The application should specify buffer sizes that allow to hold the largest packet the application expects in inbound (resp. outbound) direction. Packet that exceed the buffer size will be truncated. Too small buffers will effectively reduce the MTU in inbound/outbound direction for this application. register_app() returns OK on success. Both pointer-pointers will be set to point to the out/in buffers. The signature of the callback function looks as follows:

void callback(void* buffer, size_t packet_len, filterid_t filterid)

As a first argument the pointer to the received data will be passed. The second arguments describes the size of the received packet, and the last argument the filter id that matched this packet.

5.3.2 Application deregistration

To terminate the session with bfdmux the application has to call unregister_app() without any arguments. An OK as return value indicates a successful uncoupling from the bfdmux instance.

5.3.3 Attaching filters

While registered, applications can attach themselves to filters. Network packets matching these filters will then be forwarded to the application via the callback. To attach the application to a given filter, you have to use the attach() function with a char pointer to a filter string as the argument. The result will be a filter id greater or equal to zero. Negative return values indicate an error.

5.3.4 Detatching filters

The function detach(), which takes a filter id as the single argument, detaches the filter from the application. OK will be returned on success, ERR otherwise.

5.3.5 Sending network packets

The application should first assemble the packet it wants to send in the shared memory buffer. Then there is a function to send the data out to the network. send() just takes the length of the packet as argument.

5.4 Managing debug output

In the debug.h header file you can specify the debug level for messages on stdout. Here an overview of the different debug levels:

0: No messages will be printed on stdout.

1: Only error messages will be printed.

2: Error and information messages will be printed.

3: Information, errors and packet data as ascii will be printed.

4: Information, errors and packet data as ascii and hex will be printed.

Note:

After changing the debug level you need to rebuild libbfdmux and your application using libbfdmux.

Chapter 6

Storage of filter code

6.1 General

Operators will be stored in an array representing the operator tree of the expression using polish notation.

6.1.1 Example

((int8[0] + int8[1]) == 5) && (int16[0] >= 32))

BIC Representation:

0x42, 0x00000010 Operator &&, Subtree Size: 16 bytes

0x11 Operator ==

0x31 Operator +

0x71 Load Value: int8

0x61 Immediate Value: 1 byte

0x00 Data: 0

0x71 Load Value: int8

0x61 Immediate Value: 1 byte

0x01 Data: 1

0x61 Immediate Value: 1 byte

0x05 Data: 5

0x22 Operator >=

0x72 Load Value: int16

0x61 Immediate Value: 1 byte

0x00 Data: 0

0x61 Immediate Value: 1 byte

0x20 Data: 32

6.2 Op-Codes

6.2.1 comparison

- 0x11 Equal (==)
- 0x12 Signed greater (>)
- 0x13 Signed less (<)
- 0x14 Unsigned greater (})
- 0x15 Unsigned less ({)

- 0x21 Unequal (!=)
- 0x22 Signed greater or equal (>=)
- 0x23 Signed less or equal (<=)
- 0x24 Unsigned greater or equal (}=)
- 0x25 Unsigned less or equal ({=)

6.2.2 arithmetic

- 0x31 Addition (+)
- 0x32 Subtraction (-)
- 0x33 Multiplication (∗)
- 0x34 Integer Division (/)
- 0x35 Modulus (%)

6.2.3 logical

- 0x41 Not (!)
- 0x42 And (&&), 4 byte value for subtree size follows
- 0x43 Or (||), 4 byte value for subtree size follows

6.2.4 bitwise

- 0x51 Not (∼)
- 0x52 And (&)
- 0x53 Or (|)
- 0x54 Xor (^)

6.2.5 load data

- 0x61 Immediate value: 1 byte
- 0x62 Immediate value: 2 bytes
- 0x63 Immediate value: 4 bytes
- 0x64 Immediate value: 8 bytes
- 0x71 Indirect value: 1 byte, offset calculated in following subtree

- 0x72 Indirect value: 2 bytes, offset calculated in following subtree
- 0x73 Indirect value: 4 bytes, offset calculated in following subtree
- 0x74 Indirect value: 8 bytes, offset calculated in following subtree

Chapter 7

Libbfdmux design

7.1 Client interface protocol

The client interface protocol is specified in the bfdmux_ciprot.h header file. It offers the ability to create command packets and functions to extract meta information like the packet size and the packet command. Additionally a function to check the validity of a given command packet is implemented.

Note:

Functions relating to the command interface carry the abbreviation 'ci' in their name

7.2 Command packets

Command packets are structs where the first memeber describes the command itself. This command member is of type cmd_t. Command arguments are given using other struct members that are placed after the command member. These structs are then sent over the command socket to bfdmux. This means that struct internal padding will also be sent over the channel.

Warning:

To ensure that communication on the command interface works, you need to build libbfdmux on the same machine as bfdmux or take special care of struct member alignment.

7.2.1 Command structure

The header file contains for every command packet the accordingly struct with its command. While building the command packet you always need to set the cmd member to the command type. The command types can also be found in the header file.

E.g. the register command packet is the struct cmd_register where you need to set the .cmd member to CMD_REGISTER.

Click on the struct types for a description of the commands:

Commands sent from the application to bfdmux:

- struct cmd_register
- struct cmd_unregister
- struct cmd_attach (see Note)
- struct cmd_detach
- struct cmd_send
- struct cmd_error
- struct cmd_recv_answer

Commands sent from bfdmux to the application:

- struct cmd_register_answer
- struct cmd_unregister_answer
- struct cmd_attach_answer
- struct cmd_detach_answer
- struct cmd_send_answer
- struct cmd_recv

Note:

A special command is cmd_attach. It has a variable length because you should be able to attach the application to any given filter. The filter string length has to be written in the second struct member called len.

7.3 Command channel

To send and receive commands to/from bfdmux, libbfdmux uses unix sockets. While calling the register_app() function libbfdmux connects itself to the Unix socket located at BFDMUX_SOCK_PATH, which is defined in the bfdmux.h header file. This socket connection is closed when calling the unregister_app() function.

To send commands over this channel libbfdmux implements the bfdmux_ci_send() function which takes a pointer to a data segment and the packet length as argument. bfdmux_ci_recv() makes a blocking receive on the command socket and takes the same arguments as bfdmux_ci_send().

Libbfdmux uses two threads for sending and receiving commands. The receive-thread gets launched with the server_thread_start() function and waits for incoming commands. In case of a CMD_RECV command, the event handler function will be called with the network packet position and its size. After the handler exits a CMD_RECV_ANSWER command will be sent back to bfdmux indicating that the application is ready to receive a new network packet.

Other incoming packets will be buffered until the application itself calls the recv_answer() function. This function does busy waiting until an answer command is available, sets the pointer-pointer argument to point to the command and returns the command size.

7.4 Data channel

Network packets are sent over two shared memory segments, one for each direction. The shared memory segment gets created by the application while calling register_app(). On the registration procedure also both shared memory segment sizes and keys are transmitted to bfdmux as the arguments of the command.

7.5 Read-write locks

rwlock.c and rwlock.h implements read-write locks using semaphores. Read-write locks can be acquired as read-only or read-and-write locks. With rwlock_create() such a lock can be created, and with rwlock_destroy() the application can destroy the lock.

To acquire a lock you can call rwlock_acquire() with the lock id as first argument. The second argument describes whether you want a read-and-write or read-only lock. rwlock_acquire() is blocking. To use this function in a non blocking manner, the application can use rwlock_try_acquire() which returns true on success and false on failure. To elevate a read-only lock to a read-and-write lock you can use rwlock_elevate(). The inverse operation can be performed using rwlock_lower(). rwlock_release() finally frees that lock.

7.6 Helper functions

Libbfdmux also offers some helper function to simplify packet filter generation. These helper functions are implemented in tools.c:

- build_ipv4_udp_filter()
- build_ipv4_tcp_filter()
- build_tcp_filter()
- build_udp_filter()
- build_ipv4_filter()

These take filtering options as arguments and return a filter string that can be used with the attach() command.

Warning:

Don't forget to free the filter string after its usage!

Chapter 8

Libbfdmux internal development manual

8.1 Adding a command to libbfdmux

8.1.1 Command definition

To add custom commands to libbfdmux you have to first define the command structure in bfdmux_ciprot.h. The first struct member has to be of type cmd_t. Additionally as many members as wanted can be added. Then you have to define the command id.

The functions cmd_get(), cmd_get_size() and cmd_check() implemented in bfdmux_ciprot.c should then also be adapted to handle this command type appropriately.

8.1.2 Commands fired by libbfdmux

If the additional command should be fired by the application/libbfdmux, you have to add a new function in libbfdmux.c to build and send the command packet. To build the command packet create an instance of your new struct and set the first member to your defined command id. Then set all additional struct members and finally send the command packet using bfdmux_ci_send(), where the first argument is a void pointer to your command packet, and the second argument is the packet length.

8.1.3 Answer commands

If you intend to receive an answer you have to repeat the steps to design the answer command. Then you can add your command in the server_thread() function by inserting a new case where all other answer commands resides. To get the answer for the sent command now just call recv_answer() with a pointer-pointer as first argument. This argument will then be set to point to the answer and as a return value you will get the size of the answer command.

8.1.4 Commands fired by bfdmux

For a command that should be fired by bfdmux you again first have to design the command packet and then you can create a new case in the server_thread() switch to handle this command.

Chapter 9

Sample libbfdmux applications

9.1 Overview

- Bfdmuxchat
- Bfdmuxsniff

9.2 Bfdmuxchat

This sample application implements a chat service based on libbfdmux. Plaintext chat messages will be sent to bfdmux containing the nickname as a prefix. To differentiate between own, sent messages and messages from others, every client attaches a filter that compares the first byte against the first byte of the nickname.

9.2.1 Usage

First run the bfdmux instance and the Message Queue Loopback and then you can launch multiple chat applications. To quit the chat just hit Ctrl+C on your keyboard.

9.3 Bfdmuxsniff

By hitting the Ctrl+\ keystroke on your keyboard, you will be asked to the enter a filter. This filter will then be attached at the sniff application and metadata of all matched packets will be displayed.

To exit the application press Ctrl+C.

Bfdmuxsniff currently supports IPv4, TCP, UDP and ICMP.

9.4 Message Queue Loopback

This application implements a loopback for the message queue interface used by default by bfdmux. All outgoing packets will immediately be re-injected and a signal will be sent to bfdmux indicating a new packet at its inbound message queue.

9.4.1 Usage

To run msgq_loopback just build it using the **make** utility and then execute **./msgq_loopback**.

Note:

Please run msgq_loopback only after bfdmux. This application needs the PID of the bfdmux instance to be able to re-inject packets.

9.4.2 Troubleshooting

If you get weird output on your applications try to run **./msgq_clean** to flush all message queues used by bfdmux.

Chapter 10

Todo List

Global MAX_FILTER_CODE_SIZE Implement a better restriction for the filter processing time per application

Global MQ_FLAG Verify and change if possible.

Chapter 11

Directory Hierarchy

11.1 Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

bfdmux . . . 41
  src . . . 56
    include . . . 47
      netif . . . 51
    netif . . . 50
doc . . . 45
libbfdmux . . . 48
  bfdmuxchat . . . 42
    msgq_loopback . . . 49
      src . . . 55
  bfdmuxsniff . . . 44
    bfdmuxinject . . . 43
      src . . . 54
    src . . . 53
  src . . . 52
    include . . . 46

Chapter 12

Data Structure Index

12.1 Data Structures

Here are the data structures with brief descriptions:

client_app (Holds all information about a registered application) 57
cmd_attach (Attach command Attach a filter to the application) 59
cmd_attach_answer (Answer command to the attach command) 60
cmd_detach (Detach command) 61
cmd_detach_answer (Answer command to the detach command) 62
cmd_error (Error command) 63
cmd_recv (Receive command) 64
cmd_recv_answer (Answer command to the receive command) 65
cmd_register (Register command) 66
cmd_register_answer (Answer command to the register command) 67
cmd_send (Send command) 68
cmd_send_answer (Answer command to the send command) 69
cmd_unregister (Unregister command) 70
cmd_unregister_answer (Answer command to the unregister command) . . . 71
filter (Encapsulates filter code and it's length) 72
nic_message_buf (Message queue buffer) 73
op_def_t (Defines a type for operator definition entries) 74

Chapter 13

File Index

13.1 File List

Here is a list of all documented files with brief descriptions:

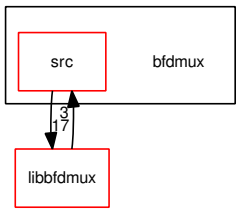
bfdmux/src/bfdmux.c (Bfdmux core functionality) 75
bfdmux/src/codegen.c (Code synthesizer for bfdmux filters) 82
bfdmux/src/filter.c (Provides high level filtering functionality to bfdmux) . . 87
bfdmux/src/opdefs.c (Bfdmux core functionality) 119
bfdmux/src/register.c (Application registration API) 122
bfdmux/src/server.c (Client application interface) 126
bfdmux/src/vm.c (Implements a virtual machine for executing compiled intermediate language byte code) 134
bfdmux/src/include/codegen.h (Code synthesizer for bfdmux filters) 90
bfdmux/src/include/filter.h (Application registration API) 93
bfdmux/src/include/netif.h (Interface file to the network card) 98
bfdmux/src/include/opdefs.h (Header file for opcode definitions) 102
bfdmux/src/include/register.h (Application registration API) 104
bfdmux/src/include/server.h (Server thread header file) 109
bfdmux/src/include/vm.h (Interface for filter execution virtual machine) . . . 113
bfdmux/src/include/netif/mqif.h (Header file for the example network interface) 101
bfdmux/src/netif/mqif.c (Sample network interface driver using two (in, out) message queues) 115
doc/**doc_bfdmux.filterlanguage.h** . . . . . **??**
doc/**doc_bfdmux.internal.development.manual.h** . . . . . **??**
doc/**doc_bfdmux.main.h** . . . . . **??**
doc/**doc_bfdmux.network.interface.manual.h** . . . . . **??**
doc/**doc_interface.msgq_loopback.h** . . . . . **??**
doc/**doc_libbfdmux.application.developer.manual.h** . . . . . **??**
doc/**doc_libbfdmux.bfdmux.bic.h** . . . . . **??**
doc/**doc_libbfdmux.design.h** . . . . . **??**
doc/**doc_libbfdmux.internal.development.manual.h** . . . . . **??**
doc/**doc_libbfdmux.sample.applications.h** . . . . . **??**

libbfdmux/bfdmuxchat/bfdmuxchat.c (Sample chat application) 137
libbfdmux/bfdmuxchat/msgq_loopback/src/msgq_clear.c (Clean all message queues) 140
libbfdmux/bfdmuxchat/msgq_loopback/src/msgq_loopback.c (Message queue loopback) 141
libbfdmux/bfdmuxsniff/bfdmuxinject/src/bfdmuxinject.c (Inject real network packets from your 'to-the-world-connected' NIC into bfdmux) 142
libbfdmux/bfdmuxsniff/src/bfdmuxsniff.c (A sniffer written for bfdmux) . . 145
libbfdmux/src/bfdmux_ciprot.c (Bfdmux client protocol interface implementation) 149
libbfdmux/src/libbfdmux.c (Interface for applications that want to use bfdmux) 178
libbfdmux/src/rwlock.c (Read write lock) 187
libbfdmux/src/tools.c (Helper functoin and additional tools used by libbfdmux) 192
libbfdmux/src/include/bfdmux.h (Bfdmux twek options) 152
libbfdmux/src/include/bfdmux_ciprot.h (Bfdmux command interface protocol header file Declaration of the available command packets and the corresponding command types) 155
libbfdmux/src/include/debug.h (Debug makro definitions) 161
libbfdmux/src/include/libbfdmux.h (Libbfdmux API) 163
libbfdmux/src/include/rwlock.h (Read/write lock header file) 168
libbfdmux/src/include/tools.h (Header file for helper and additional functions) 173

Chapter 14

Directory Documentation

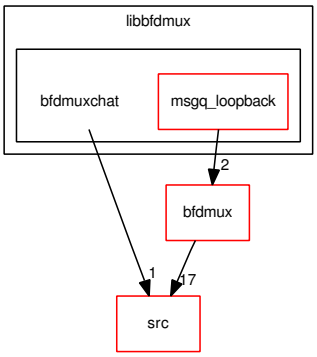
14.1 bfdmux/ Directory Reference



Directories

- directory src

14.2 libbfdmux/bfdmuxchat/ Directory Reference



Directories

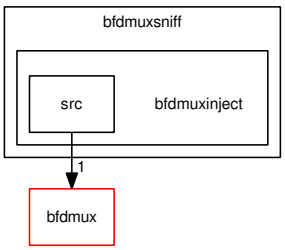
- directory msgq_loopback

Files

- file bfdmuxchat.c

  Sample chat application.

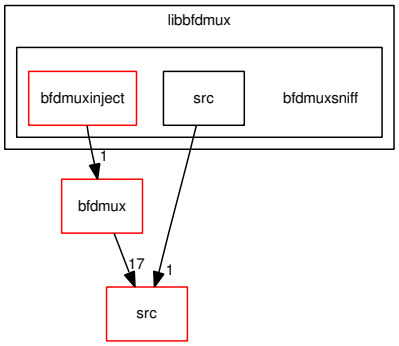
14.3 libbfdmux/bfdmuxsniff/bfdmuxinject/ Directory Reference



Directories

- directory src

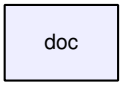
14.4 libbfdmux/bfdmuxsniff/ Directory Reference



Directories

- directory bfdmuxinject
- directory src

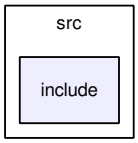
14.5 doc/ Directory Reference



Files

- file **doc_bfdmux.filterlanguage.h**
- file **doc_bfdmux.internal.development.manual.h**
- file **doc_bfdmux.main.h**
- file **doc_bfdmux.network.interface.manual.h**
- file **doc_interface.msgq_loopback.h**
- file **doc_libbfdmux.application.developer.manual.h**
- file **doc_libbfdmux.bfdmux.bic.h**
- file **doc_libbfdmux.design.h**
- file **doc_libbfdmux.internal.development.manual.h**
- file **doc_libbfdmux.sample.applications.h**

14.6 libbfdmux/src/include/ Directory Reference



Files

- file bfdmux.h

  Bfdmux twek options.

- file bfdmux_ciprot.h

  Bfdmux command interface protocol header file Declaration of the available command packets and the corresponding command types.

- file debug.h

  Debug makro definitions.

- file libbfdmux.h

  Libbfdmux API.

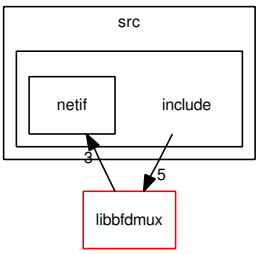
- file rwlock.h

  Read/write lock header file.

- file tools.h

  Header file for helper and additional functions.

14.7 bfdmux/src/include/ Directory Reference



Directories

- directory netif

Files

- file codegen.h

  Code synthesizer for bfdmux filters.

- file filter.h

  Application registration API.

- file netif.h

  Interface file to the network card.

- file opdefs.h

  Header file for opcode definitions.

- file register.h

  Application registration API.

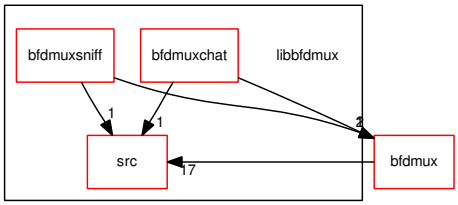
- file server.h

  Server thread header file.

- file vm.h

  Interface for filter execution virtual machine.

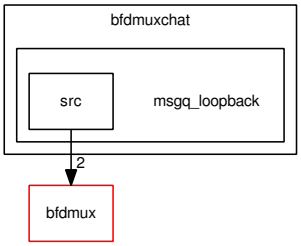
14.8 libbfdmux/ Directory Reference



Directories

- directory bfdmuxchat
- directory bfdmuxsniff
- directory src

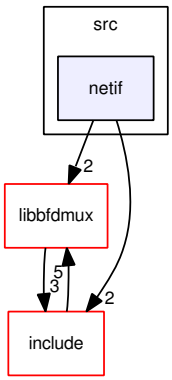
14.9 libbfdmux/bfdmuxchat/msgq_loopback/ Directory Reference



Directories

- directory src

14.10 bfdmux/src/netif/ Directory Reference

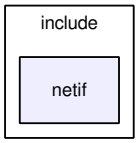


Files

- file mqif.c

  Sample network interface driver using two (in, out) message queues.

14.11 bfdmux/src/include/netif/ Directory Reference

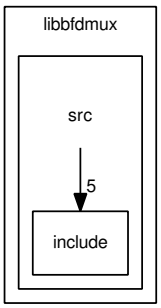


Files

- file mqif.h

  Header file for the example network interface.

14.12 libbfdmux/src/ Directory Reference



Directories

- directory include

Files

- file bfdmux_ciprot.c

  Bfdmux client protocol interface implementation.

- file libbfdmux.c

  Interface for applications that want to use bfdmux.

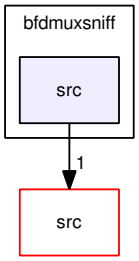
- file rwlock.c

  Read write lock.

- file tools.c

  Helper functoin and additional tools used by libbfdmux.

14.13 libbfdmux/bfdmuxsniff/src/ Directory Reference

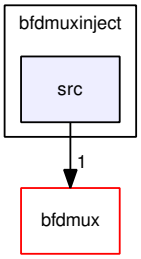


Files

- file bfdmuxsniff.c

  A sniffer written for bfdmux.

14.14 libbfdmux/bfdmuxsniff/bfdmuxinject/src/ Directory Reference

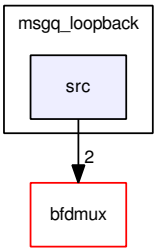


Files

- file bfdmuxinject.c

  Inject real network packets from your 'to-the-world-connected' NIC into bfdmux.

14.15 libbfdmux/bfdmuxchat/msgq_loopback/src/ Directory Reference



Files

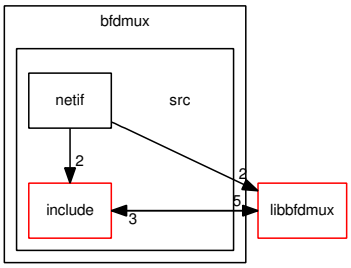
- file msgq_clear.c

  Clean all message queues.

- file msgq_loopback.c

  Message queue loopback.

14.16 bfdmux/src/ Directory Reference



Directories

- directory include
- directory netif

Files

- file bfdmux.c

  Bfdmux core functionality.

- file codegen.c

  Code synthesizer for bfdmux filters.

- file filter.c

  Provides high level filtering functionality to bfdmux.

- file opdefs.c

  Bfdmux core functionality.

- file register.c

  Application registration API.

- file server.c

  Client application interface.

- file vm.c

  Implements a virtual machine for executing compiled intermediate language byte code.

Chapter 15

Data Structure Documentation

15.1 client_app Struct Reference

Holds all information about a registered application.

Data Fields

- int num_filters
  Length of the filter array.
- sock_t command_socket
  Handle of the command connection socket.
- size_t size_in
  Size of the shared memory buffer for packets towards the application.
- size_t size_out
  Size of the shared memory buffer for packets going to be sent over the NIC.
- void * buf_in
  Pointer to shared memory for packets to the application.
- void * buf_out
  Pointer to shared memory for packets to the NIC.
- bool can_receive
  Indicates that the application buffer can receive a packet.
- struct filter * filters
  An array of filters.

15.1.1 Detailed Description

Holds all information about a registered application.

Definition at line 35 of file register.h.

15.1.2 Field Documentation

15.1.2.1 void* client_app::buf_in

Pointer to shared memory for packets to the application.

This pointer might be NULL for newly connected applications!

Definition at line 45 of file register.h.

15.1.2.2 void* client_app::buf_out

Pointer to shared memory for packets to the NIC.

This pointer might be NULL for newly connected applications!

Definition at line 50 of file register.h.

15.1.2.3 bool client_app::can_receive

Indicates that the application buffer can receive a packet.

Set to false whenever buf_in contains unread data. Will be reset to true, after the application read and confirmed the incoming packet. If the application does not read the packet from the buffer early enough, packets to arriving for this application will be dropped.

Definition at line 55 of file register.h.

15.1.2.4 struct filter* client_app::filters [read]

An array of filters.

Some elements might have the code member of the filter struct set to NULL. These are no valid filters. They are kept to ensure the filter indices reported to the application are valid array indices.

Definition at line 61 of file register.h.

15.2 cmd_attach Struct Reference

Attach command Attach a filter to the application.

Data Fields

- cmd_t cmd
  Command type (should be CMD_ATTACH).
- size_t len
  Length of the filter string.
- char filter [1]
  Filter string.

15.2.1 Detailed Description

Attach command Attach a filter to the application.

Warning:

This command has a variable length because of the filter string.

Definition at line 99 of file bfdmux_ciprot.h.

15.3 cmd_attach_answer Struct Reference

Answer command to the attach command.

Data Fields

- cmd_t cmd

  Command type (should be CMD_ATTACH_ANSWER).

- filterid_t filter_id

  Filter ID of the attached filter.

15.3.1 Detailed Description

Answer command to the attach command.

Definition at line 111 of file bfdmux_ciprot.h.

15.3.2 Field Documentation

15.3.2.1 filterid_t cmd_attach_answer::filter_id

Filter ID of the attached filter.

Will be -1 if bfdmux was not able to attach the filter.

Definition at line 115 of file bfdmux_ciprot.h.

15.4 cmd_detach Struct Reference

Detach command.

Data Fields

- cmd_t cmd

  Command type (should be CMD_DETACH).

- filterid_t filter_id

  Filter ID of the filter the application wants to detach.

15.4.1 Detailed Description

Detach command.

Detach a filter from the applicatoin

Definition at line 127 of file bfdmux_ciprot.h.

15.5 cmd_detach_answer Struct Reference

Answer command to the detach command.

Data Fields

- cmd_t cmd

  Command type (should be CMD_DETACH_ANSWER).

15.5.1 Detailed Description

Answer command to the detach command.

Definition at line 137 of file bfdmux_ciprot.h.

15.6 cmd_error Struct Reference

Error command.

Data Fields

- cmd_t cmd

  Command type (should be CMD_ERROR).

15.6.1 Detailed Description

Error command.

This command will be used in both ways (bfdmux to application, application to bfdmux) and describes an error to the previous sent command.

Definition at line 192 of file bfdmux_ciprot.h.

15.7 cmd_recv Struct Reference

Receive command.

Data Fields

- cmd_t cmd
  Command type (should be CMD_RECV).
- size_t len
  Size of packet that arrived.
- filterid_t filter_id
  Id of filter that matched to the packet.

15.7.1 Detailed Description

Receive command.

Command to inform an application about incoming data

Definition at line 169 of file bfdmux_ciprot.h.

15.8 cmd_recv_answer Struct Reference

Answer command to the receive command.

Data Fields

- cmd_t cmd

  Command type (should be CMD_RECV_ANSWER).

15.8.1 Detailed Description

Answer command to the receive command.

Definition at line 181 of file bfdmux_ciprot.h.

15.9 cmd_register Struct Reference

Register command.

Data Fields

- cmd_t cmd
  Command type (should be CMD_REGISTER).
- smkey_t key_in
  Shared memory segment key of the inbound buffer (world to application).
- size_t size_in
  Size of the inbound buffer.
- smkey_t key_out
  Shared memory segment key of the outbound buffer (application to world).
- size_t size_out
  Size of the outbound buffer.

15.9.1 Detailed Description

Register command.

Definition at line 57 of file bfdmux_ciprot.h.

15.10 cmd_register_answer Struct Reference

Answer command to the register command.

Data Fields

- cmd_t cmd

  Command type (should be CMD_REGISTER_ANSWER).

15.10.1 Detailed Description

Answer command to the register command.

Definition at line 73 of file bfdmux_ciprot.h.

15.11 cmd_send Struct Reference

Send command.

Data Fields

- cmd_t cmd

  Command type (should be CMD_SEND).

- size_t len

  Number of bytes the application wants to send.

15.11.1 Detailed Description

Send command.

After sending data out (by copying to the outbound shared memory segment), the application needs to inform bfdmux

Definition at line 147 of file bfdmux_ciprot.h.

15.12 cmd_send_answer Struct Reference

Answer command to the send command.

Data Fields

- cmd_t cmd

  Command type (should be CMD_SEND_ANSWER).

- size_t len

  Number of bytes bfdmux was able to send to the world.

15.12.1 Detailed Description

Answer command to the send command.

Definition at line 157 of file bfdmux_ciprot.h.

15.13 cmd_unregister Struct Reference

Unregister command.

Data Fields

- cmd_t cmd

  Command type (should be CMD_UNREGISTER).

15.13.1 Detailed Description

Unregister command.

Definition at line 81 of file bfdmux_ciprot.h.

15.14 cmd_unregister_answer Struct Reference

Answer command to the unregister command.

Data Fields

- cmd_t cmd

  Command type (should be CMD_UNREGISTER_ANSWER).

15.14.1 Detailed Description

Answer command to the unregister command.

Definition at line 89 of file bfdmux_ciprot.h.

15.15 filter Struct Reference

Encapsulates filter code and it's length.

Data Fields

- uint8_t * code

  A pointer to a memory location where the virtual byte code resides.

- int len

  The length in bytes of the byte code.

15.15.1 Detailed Description

Encapsulates filter code and it's length.

Definition at line 25 of file register.h.

15.16 nic_message_buf Struct Reference

Message queue buffer.

Data Fields

- long mtype

  Type of the message (is always > 0).

- char mtext [NIC_MQ_SIZE]

  Payload.

15.16.1 Detailed Description

Message queue buffer.

This message queue struct is used to send data from the NIC to the world and to receive data from the world at the NIC.

Definition at line 26 of file mqif.h.

15.17 op_def_t Struct Reference

Defines a type for operator definition entries.

Data Fields

- char opstr [MAX_OPERATOR_STRING_LENGTH]

  The string representing the operator.

- uint8_t opcode

  The binary opcode the operator maps to.

- uint8_t reserved_length

  The number of bytes that should be reserved for this operator. Usually this is exactly one byte. See c file for exceptions.

- uint8_t arity

  Specifies if the operator expects left, right, or both sides to be operands. 0x10 for left-unary, 0x01 for right-unary, 0x11 for binary operators.

15.17.1 Detailed Description

Defines a type for operator definition entries.

Warning:

Operator strings cannot contain brackets!

Definition at line 25 of file opdefs.h.

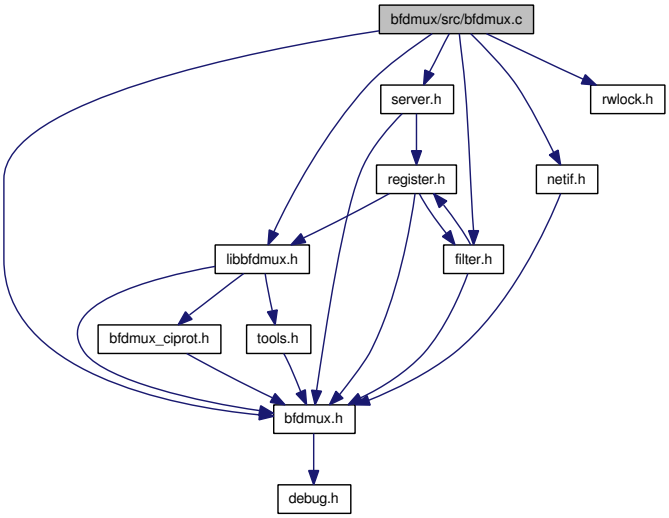
Chapter 16

File Documentation

16.1 bfdmux/src/bfdmux.c File Reference

Bfdmux core functionality.

Include dependency graph for bfdmux.c:



Defines

- #define SERVER_THREAD_PRIORITY 1000

  Server thread priority.

Functions

- void wait_for_new_packet_and_lock (void)

  Waits until the ring buffer contains new data for demultiplexing.

- bool check_demuxer_idle_and_lock (void)

  Checks whether the demultiplexer can take a new processing request.

- bool demux (void *data, int len)

  Tries to demux the given packet and forward it to the application.

- void quit (int signum)

  Signal handler for SIGINT to catch Ctrl+C on stdin.

- int main ()

  Main function that initializes bfdmux and starts the server thread to allow applications to connect.

Variables

- struct sigaction quitsa

  Signal handler action structure to catch Ctrl+C on stdin.

- pthread_t server_thread

  Handle of server thread.

- sock_t server_socket

  Listen socket for client command connections.

- int queue_lock

  Rwlock handle that should be acquired to access the packet processing queue.

- struct {
  } queue [PROC_QUEUE_LEN]

  Ring buffer of pending demultiplexing requests
  .

- int queue_first

  Index of first valid entry in the queue ring buffer.

- int queue_len

  Number of valid entries, starting from index queue_first in queue.

16.1.1 Detailed Description

Bfdmux core functionality.

This is the bfdmux main thread. It spawns the client application server thread and takes care of the actual packet filtering and forwarding.

Definition in file bfdmux.c.

16.1.2 Function Documentation

16.1.2.1 bool check_demuxer_idle_and_lock (void)

Checks whether the demultiplexer can take a new processing request.

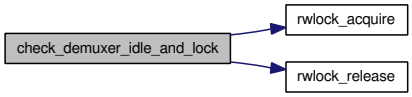
If the demuxer can accept a new request, the function leaves the queue_lock locked for reading, such that the caller can add it's demux request right away.

Warning:

The release of the read lock is left to the caller!

Definition at line 103 of file bfdmux.c.

Here is the call graph for this function:



Here is the caller graph for this function:

16.1.2.2 bool demux (void * *data*, int *len*)

Tries to demux the given packet and forward it to the application.

Parameters:

data Points to the packet in memory

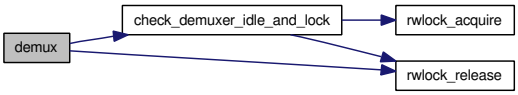
len Length of the packet data in bytes

Returns:

true if the packet could be put into the demuxer's queue; otherwise false.

Definition at line 133 of file bfdmux.c.

Here is the call graph for this function:



Here is the caller graph for this function:

16.1.2.3 int main ()

Main function that initializes bfdmux and starts the server thread to allow applications to connect.

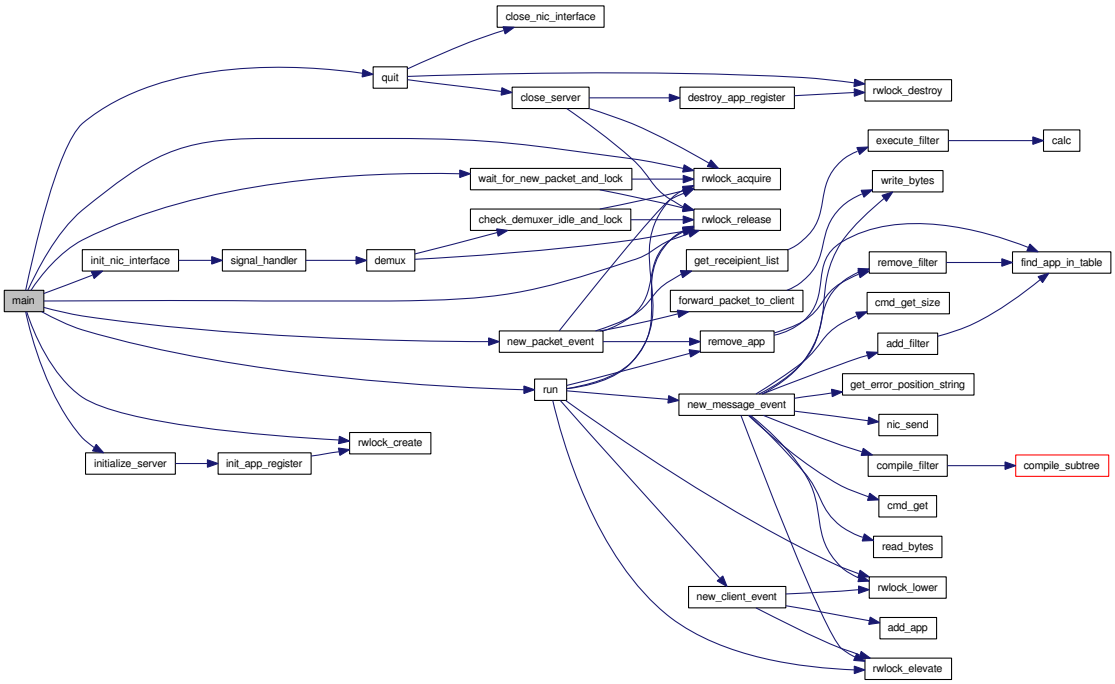
This represents the main entry point of bfdmux. After initialization, this thread enters a packet processing loop. The loop can be interrupted by the SIGINT signal.

Returns:

always returns zero

Definition at line 184 of file bfdmux.c.

Here is the call graph for this function:



16.1.2.4 void quit (int *signum*)

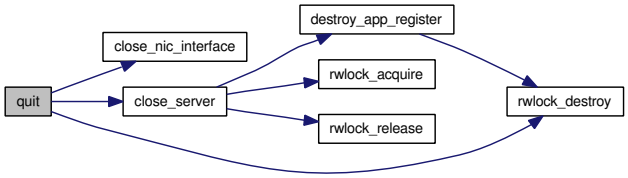
Signal handler for SIGINT to catch Ctrl+C on stdin.

Parameters:

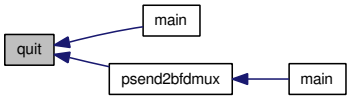
signum The signal number the handler was invoked for

Definition at line 159 of file bfdmux.c.

Here is the call graph for this function:



Here is the caller graph for this function:



16.1.2.5 void wait_for_new_packet_and_lock (void)

Waits until the ring buffer contains new data for demultiplexing.

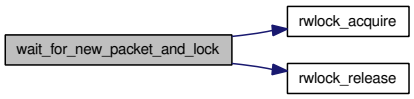
If a new processing request has been found, the rwlock queue_lock will be left locked for reading, such that the caller can continue right away.

Warning:

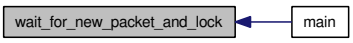
The release of the queue_lock is left to the caller upon return!

Definition at line 69 of file bfdmux.c.

Here is the call graph for this function:



Here is the caller graph for this function:



16.1.3 Variable Documentation

16.1.3.1 struct { ... } queue[PROC_QUEUE_LEN]

Ring buffer of pending demultiplexing requests

.

packet_data Points to a packet in memory that still needs to be processed and forwarded to the applications

packet_length Specifies the size of the packet in bytes

16.1.3.2 int queue_first

Index of first valid entry in the queue ring buffer.

Note:

The item is invalid, if queue_len is equal to zero!

Definition at line 57 of file bfdmux.c.

16.1.3.3 void * server_thread

Handle of server thread.

Server thread function.

The server thread waits for incoming commands or answers from bfdmux and handles them separately.

Parameters:

ptr NULL

Returns:

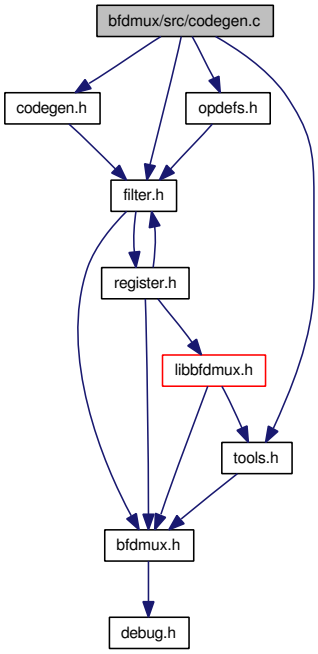
NULL

Definition at line 36 of file bfdmux.c.

16.2 bfdmux/src/codegen.c File Reference

Code synthesizer for bfdmux filters.

Include dependency graph for codegen.c:



Functions

- int substrfind (char *str, int start_pos, int end_pos, char *substr)

  Searches for 'substr' in 'str', between 'start_pos' and 'end_pos' inclusive, and not inside brackets!

- bool ensure_enough_space (uint8_t **out, int *out_len, int *out_sz, int space_needed)

  Checks if 'out' still has 'space_needed' empty bytes.

- int find_operator (char *expr, int from_pos, int to_pos, int *pos)

  Tries to find the lowest precedence operator in an interval of 'expr'.

- void remove_spaces_and_braces (char *expr, int *from_pos, int *to_pos)

  Removes leading and trailing spaces, and brackets that sourround the expression.

- int compile_subtree (char *expr, int from_pos, int to_pos, uint8_t **out, int *out_len, int *out_sz)

  Compiles a whole subexpression and appends the byte code to 'out'.

- void compile_filter (char *expression, uint8_t **filter_code, int *filter_len)

  Compiles a filter expression in Bfdmux Filter Language into Bfdmux Intermediate Code.

16.2.1 Detailed Description

Code synthesizer for bfdmux filters.

This file provides functions to create byte code in Bfdmux Intermediate Language from filter strings. This byte code can be executed using the functions in vm.c to filter network packets.

Definition in file codegen.c.

16.2.2 Function Documentation

16.2.2.1 void compile_filter (char * *expression*, uint8_t ** *filter_code*, int * *filter_len*)

Compiles a filter expression in Bfdmux Filter Language into Bfdmux Intermediate Code.

Parameters:

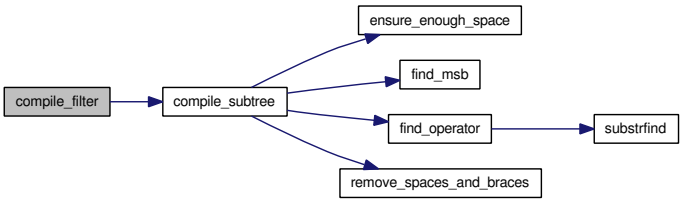
expression the expression to compile

→ ***filter_code*** Points to the memory buffer that contains the compiled filter, or NULL, if an error occurred

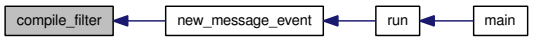
→ ***filter_len*** Indicates the length of the compiled code, or contains the error position in the filter string on failure

Definition at line 311 of file codegen.c.

Here is the call graph for this function:



Here is the caller graph for this function:



16.2.2.2 int compile_subtree (char * *expr*, int *from_pos*, int *to_pos*, uint8_t ** *out*, int * *out_len*, int * *out_sz*)

Compiles a whole subexpression and appends the byte code to 'out'.

Parameters:

expr the expression to work with

from_pos the first character of the subexpression to compile

to_pos the last character of the subexpression to compile

out a pointer to the array that holds the compiled byte code

out_len the number of bytes already in the array

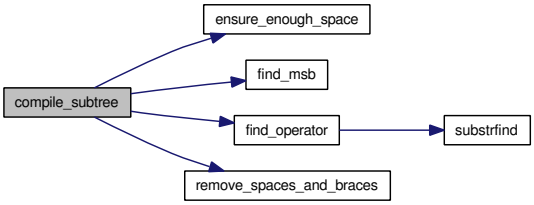
out_sz the current size of the array (reallocation, if full)

Returns:

returns 0 on success, -1 on memory error, or an index of a character in the subexpression that failed to compile

Definition at line 203 of file codegen.c.

Here is the call graph for this function:



Here is the caller graph for this function:

16.2.2.3 bool ensure_enough_space (uint8_t ** *out*, int * *out_len*, int * *out_sz*, int *space_needed*) [inline]

Checks if 'out' still has 'space_needed' empty bytes.

Parameters:

out the byte array

out_len the number of bytes already in the array 'out'

out_sz the length of the currently reserved space of 'out'

space_needed the number of bytes that need to be appended to 'out'

Returns:

returns true if enough space or reservation of larger space succeeded, otherwise false

Definition at line 72 of file codegen.c.

Here is the caller graph for this function:

16.2.2.4 int find_operator (char * *expr*, int *from_pos*, int *to_pos*, int * *pos*)

Tries to find the lowest precedence operator in an interval of 'expr'.

Parameters:

expr the expression to search in

from_pos the first character of the serach interval

to_pos the last character of the search interval

pos the position where the operator has been found, if any

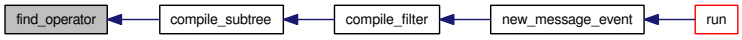
Returns:

the index of the operator in the op_list or -1 if no operator found

Definition at line 99 of file codegen.c.

Here is the call graph for this function:

Here is the caller graph for this function:



16.2.2.5 void remove_spaces_and_braces (char * *expr*, int * *from_pos*, int * *to_pos*)

Removes leading and trailing spaces, and brackets that sourround the expression.

Parameters:

expr the expression to work with

from_pos the start of the inverval to consider; may be shifted to the right

to_pos the end of the interval to consider; may be shifted left

Definition at line 122 of file codegen.c.

Here is the caller graph for this function:

16.2.2.6 int substrfind (char * *str*, int *start_pos*, int *end_pos*, char * *substr*) [inline]

Searches for 'substr' in 'str', between 'start_pos' and 'end_pos' inclusive, and not inside brackets!

Parameters:

str the string to be searched

start_pos the index of the first character of the search interval

end_pos the index of the last character of the search interval

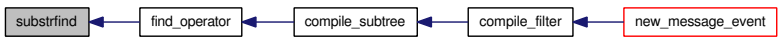
substr the substring to look for

Returns:

returns the position at which 'substr' has been found, -1 if not found, -2 on bracket error

Definition at line 36 of file codegen.c.

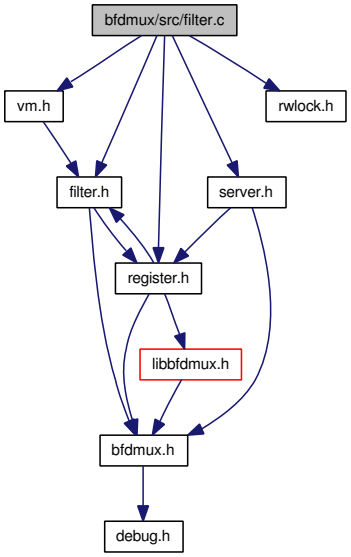
Here is the caller graph for this function:



16.3 bfdmux/src/filter.c File Reference

Provides high level filtering functionality to bfdmux.

Include dependency graph for filter.c:



Functions

- int get_receipient_list (struct client_app **app_table, int app_cnt, uint8_t *packet_data, int packet_len, struct client_app ***hits, filterid_t **fids)

  Filters a packet with all filters of all applications.

- void new_packet_event (void *packet, int len)

  Event handler for new packet data.

16.3.1 Detailed Description

Provides high level filtering functionality to bfdmux.

Definition in file filter.c.

16.3.2 Function Documentation

16.3.2.1 int get_receipient_list (struct client_app ** *app_table*, int *app_cnt*, uint8_t * *packet_data*, int *packet_len*, struct client_app *** *hits*, filterid_t ** *fids*)

Filters a packet with all filters of all applications.

Parameters:

app_table The list of applications that are registered

app_cnt The number of applications in the list

packet_data Pointer to the raw packet data

packet_len Length of packet data in bytes

→ ***hits*** Points to an array of applications that had a least one matching filter. Should be pointing to a NULL pointer initially.

→ ***fids*** Points to an array of the filter id of each application that matched the packet. Should be pointing to a NULL pointer initially.

Returns:

Number of applications with matching filters (length of 'hits'). If zero, hits has not been modified and might be invalid.

Definition at line 31 of file filter.c.

Here is the call graph for this function:

Here is the caller graph for this function:

16.3.2.2 void new_packet_event (void * *packet*, int *len*)

Event handler for new packet data.

This function will be called by bfdmux, whenever a new data packet has arrived. It invokes all filters on the packet data and tries to copy the packet to the application input buffers, if one of the applications filters matched.

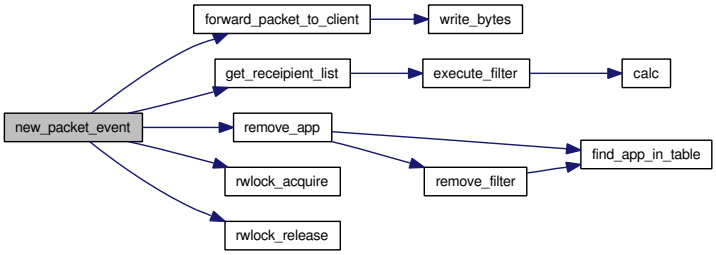
Parameters:

packet pointer to packet data

len length of packet data in bytes

Definition at line 132 of file filter.c.

Here is the call graph for this function:

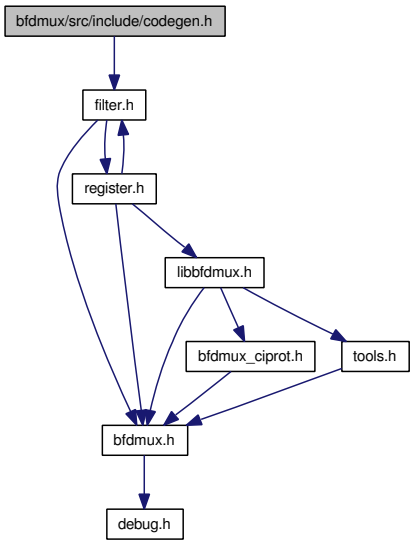


Here is the caller graph for this function:

16.4 bfdmux/src/include/codegen.h File Reference

Code synthesizer for bfdmux filters.

Include dependency graph for codegen.h:



Defines

- #define MAX_FILTER_CODE_SIZE 256

  Maximum number of bytes for a compiled filter.

- #define INITIAL_ALLOC_SIZE 64
- #define INCREMENTAL_ALLOC_SIZE 64

Functions

- void compile_filter (char *expression, uint8_t **filter_code, int *filter_len)

  Compiles a filter expression in Bfdmux Filter Language into Bfdmux Intermediate Code.

16.4.1 Detailed Description

Code synthesizer for bfdmux filters.

This file provides the interface for the filter code generator.

Definition in file codegen.h.

16.4.2 Define Documentation

16.4.2.1 #define INCREMENTAL_ALLOC_SIZE 64

Size of realloc'ed blocks if filter code doesn't fit

Definition at line 22 of file codegen.h.

16.4.2.2 #define INITIAL_ALLOC_SIZE 64

Size of initially allocated filter code block

Definition at line 21 of file codegen.h.

16.4.2.3 #define MAX_FILTER_CODE_SIZE 256

Maximum number of bytes for a compiled filter.

This is used to limit bfdmux's workload in some way.

Todo

Implement a better restriction for the filter processing time per application

Definition at line 19 of file codegen.h.

16.4.3 Function Documentation

16.4.3.1 void compile_filter (char * *expression*, uint8_t ** *filter_code*, int * *filter_len*)

Compiles a filter expression in Bfdmux Filter Language into Bfdmux Intermediate Code.

Parameters:

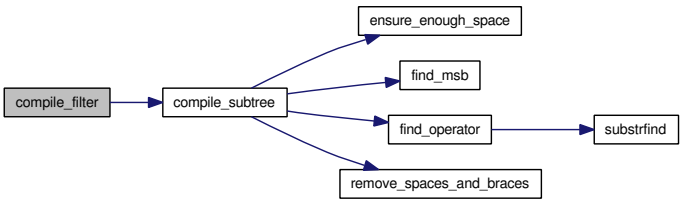
expression the expression to compile

→ ***filter_code*** Points to the memory buffer that contains the compiled filter, or NULL, if an error occurred

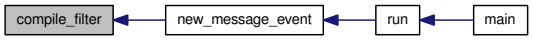
→ ***filter_len*** Indicates the length of the compiled code, or contains the error position in the filter string on failure

Definition at line 311 of file codegen.c.

Here is the call graph for this function:



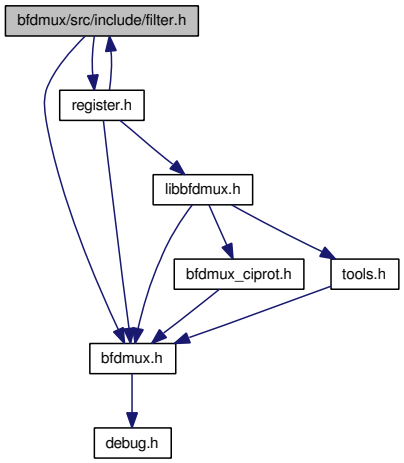
Here is the caller graph for this function:



16.5 bfdmux/src/include/filter.h File Reference

Application registration API.

Include dependency graph for filter.h:



Defines

- #define OP_EQUAL 0x11

  Operator ==.

- #define OP_SGREATER 0x12

  Operator > *(signed).*

- #define OP_SLESS 0x13

  Operator < *(signed).*

- #define OP_UGREATER 0x14

  Operator > *(unsigned).*

- #define OP_ULESS 0x15

  Operator < *(unsigned).*

- #define OP_UNEQUAL 0x21

  Operator !=.

- #define OP_SGREATEREQUAL 0x22

  Operator >= *(signed).*

- #define OP_SLESSEQUAL 0x23

  Operator <= *(signed).*

- #define OP_UGREATEREQUAL 0x24
  Operator >= (unsigned).
- #define OP_ULESSEQUAL 0x25
  Operator <= (unsigned).
- #define OP_ADD 0x31
  Operator +.
- #define OP_SUB 0x32
  Operator -.
- #define OP_MULT 0x33
  Operator ∗.
- #define OP_IDIV 0x34
  Operator / (integer division).
- #define OP_MOD 0x35
  Operator %.
- #define OP_NOT 0x41
  Operator !
- #define OP_AND 0x42
  Operator &&.
- #define OP_OR 0x43
  Operator ||.
- #define OP_BNOT 0x51
  Operator ∼.
- #define OP_BAND 0x52
  Operator &.
- #define OP_BOR 0x53
  Operator |.
- #define OP_BXOR 0x54
  Operator ^.
- #define OP_INT8 0x61
  8 bit immediate value, data follows

- #define OP_INT16 0x62

  16 bit immediate value, data follows

- #define OP_INT32 0x63

  32 bit immediate value, data follows

- #define OP_INT64 0x64

  64 bit immediate value, data follows

- #define OP_LOAD8 0x71

  8 bit indirect value, location follows

- #define OP_LOAD16 0x72

  16 bit indirect value, location follows

- #define OP_LOAD32 0x73

  32 bit indirect value, location follows

- #define OP_LOAD64 0x74

  64 bit indirect value, location follows

Functions

- int get_receipient_list (struct client_app **app_table, int app_cnt, uint8_t *packet_data, int packet_len, struct client_app ***hits, filterid_t **fids)

  Filters a packet with all filters of all applications.

- void new_packet_event (void *packet, int len)

  Event handler for new packet data.

16.5.1 Detailed Description

Application registration API.

This file contains defintitions for the bfdmux application register.

Definition in file filter.h.

16.5.2 Define Documentation

16.5.2.1 #define OP_AND 0x42

Operator &&.

Expects an additional 32bit word before the two operands holding the code size of the first operand subtree in bytes. This is used to speed up filter execution when the first operand already determines the result of the operation, e.g. false && something.

Definition at line 54 of file filter.h.

16.5.2.2 #define OP_OR 0x43

Operator ||.

Expects an additional 32bit word before the two operands holding the code size of the first operand subtree in bytes. This is used to speed up filter execution when the first operand already determines the result of the operation, e.g. true || something.

Definition at line 63 of file filter.h.

16.5.3 Function Documentation

16.5.3.1 int get_receipient_list (struct client_app ** *app_table*, int *app_cnt*, uint8_t * *packet_data*, int *packet_len*, struct client_app *** *hits*, filterid_t ** *fids*)

Filters a packet with all filters of all applications.

Parameters:

app_table The list of applications that are registered

app_cnt The number of applications in the list

packet_data Pointer to the raw packet data

packet_len Length of packet data in bytes

→ ***hits*** Points to an array of applications that had a least one matching filter. Should be pointing to a NULL pointer initially.

→ ***fids*** Points to an array of the filter id of each application that matched the packet. Should be pointing to a NULL pointer initially.

Returns:

Number of applications with matching filters (length of 'hits'). If zero, hits has not been modified and might be invalid.

Definition at line 31 of file filter.c.

Here is the call graph for this function:

Here is the caller graph for this function:

16.5.3.2 void new_packet_event (void * *packet*, int *len*)

Event handler for new packet data.

This function will be called by bfdmux, whenever a new data packet has arrived. It invokes all filters on the packet data and tries to copy the packet to the application input buffers, if one of the applications filters matched.

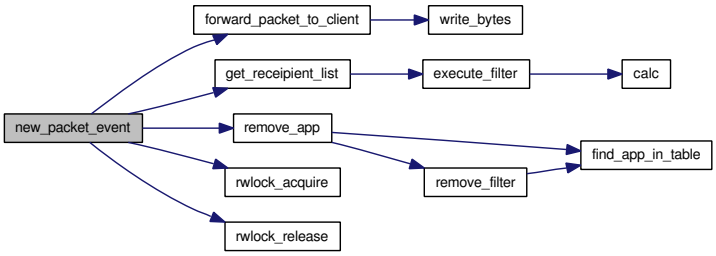
Parameters:

packet pointer to packet data

len length of packet data in bytes

Definition at line 132 of file filter.c.

Here is the call graph for this function:

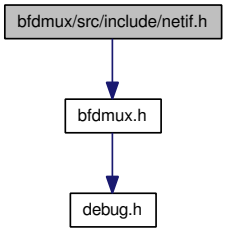


Here is the caller graph for this function:

16.6 bfdmux/src/include/netif.h File Reference

Interface file to the network card.

Include dependency graph for netif.h:



Functions

- err_t init_nic_interface ()

  Initialize the network card interface.

- err_t close_nic_interface ()

  Close the network driver.

- err_t nic_send (void *buff, size_t len)

  Send packets out to the world on the outbound message queue.

16.6.1 Detailed Description

Interface file to the network card.

This file must be included from your own network card implementation. It defines global functions that get used by other parts of the project.

Definition in file netif.h.

16.6.2 Function Documentation

16.6.2.1 err_t close_nic_interface ()

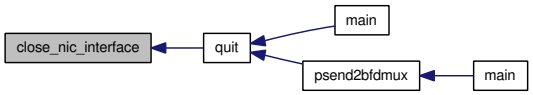
Close the network driver.

Returns:

ERR_OK on success

Definition at line 92 of file mqif.c.

Here is the caller graph for this function:



16.6.2.2 err_t init_nic_interface ()

Initialize the network card interface.

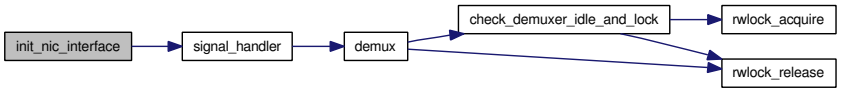
Set up signal handler and initialize the message queue to receive packets from the NIC.

Returns:

ERR_OK on success, ERR_FATAL on a fatal error.

Definition at line 51 of file mqif.c.

Here is the call graph for this function:



Here is the caller graph for this function:

16.6.2.3 err_t nic_send (void * *buff*, size_t *len*)

Send packets out to the world on the outbound message queue.

Parameters:

buff Pointer to the data array

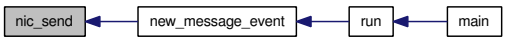
len Number of bytes to send starting at 'buff'

Returns:

ERR_OK on success, ERR_FATAL on error

Definition at line 144 of file mqif.c.

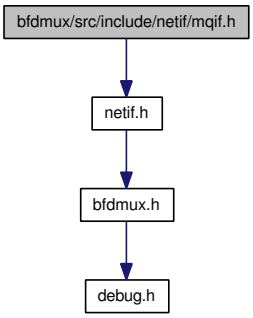
Here is the caller graph for this function:



16.7 bfdmux/src/include/netif/mqif.h File Reference

Header file for the example network interface.

Include dependency graph for mqif.h:



Data Structures

- struct nic_message_buf
  Message queue buffer.

Defines

- #define BFDMUXPIDFILE "/tmp/.bfdmux.pid"
  Location of the PID file of bfdmux.
- #define SIGNICINTERRUPT 29
  Signal number to wake up the NIC.
- #define NIC_OUT_MQ_KEY 0x9abcdef0;
  Outbound message queue key.
- #define NIC_IN_MQ_KEY 0x12345678;
  Inbound message queue key.
- #define NIC_MQ_SIZE 1500
  Message queue size.

16.7.1 Detailed Description

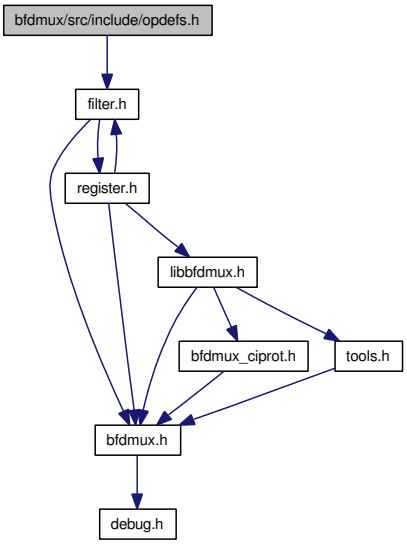
Header file for the example network interface.

Definition in file mqif.h.

16.8 bfdmux/src/include/opdefs.h File Reference

Header file for opcode definitions.

Include dependency graph for opdefs.h:



Data Structures

- struct op_def_t

  Defines a type for operator definition entries.

Defines

- #define MAX_OPERATOR_STRING_LENGTH 9

  Maximum length of an operator string in characters.

Variables

- op_def_t op_list [ ]

  List of operators and opcodes.

16.8.1 Detailed Description

Header file for opcode definitions.

Definition in file opdefs.h.

16.8.2 Variable Documentation

16.8.2.1 op_def_t op_list[ ]

List of operators and opcodes.

Operators with lower indices have lower precedence.

Warning:

If one operator string is contained in another one, the longer opstring needs to reside in a lower index in the array! Example: "<" and "<=". If this is not the case the string "4 <= 5" will be split into "4" < "= 5", which will lead to a compile error in the right subexpression!

Note:

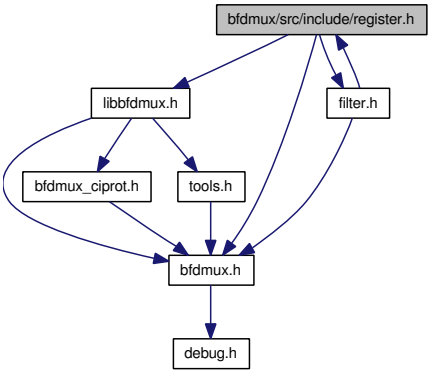
For OP_OR and OP_AND
5 specifies that 4 extra bytes should be left after the opcode: they will hold the left subtree length for skipping the right subtree evaluation if possible.
The empty operator signals the end of the operator list, do not remove it!

Definition at line 21 of file opdefs.c.

16.9 bfdmux/src/include/register.h File Reference

Application registration API.

Include dependency graph for register.h:



Data Structures

- struct filter

  Encapsulates filter code and it's length.

- struct client_app

  Holds all information about a registered application.

Functions

- void init_app_register (void)

  Initializes the application register app_table.

- void destroy_app_register (void)

  Cleans up application register app_table.

- void add_app (sock_t command_socket, void *shmaddr_in, size_t shmsize_in, void *shmaddr_out, size_t shmsize_out)

  Adds an application to the application list.

- void remove_app (sock_t command_socket)

  Removes an application from the application list.

- filterid_t add_filter (sock_t command_socket, uint8_t *filter_code, int filter_len)

  Adds a filter for a specific application (identified by it's socket).

- void remove_filter (sock_t command_socket, filterid_t filterid)

  Removes a filter for an application.

- int find_app_in_table (struct client_app **table, int table_len, int command_socket)

  Searches for the application's index given it's command socket handle.

Variables

- struct client_app ** app_table

  Global table of connected client applications.

- int num_apps

  Length of app_table.

- int app_table_lock

  An rwlock handle that should be acquired before reading from or writing to the application list.

16.9.1 Detailed Description

Application registration API.

This file contains defintitions for the bfdmux application register.

Definition in file register.h.

16.9.2 Function Documentation

16.9.2.1 void add_app (sock_t *command_socket*, void * *shmaddr_in*, size_t *shmsize_in*, void * *shmaddr_out*, size_t *shmsize_out*)

Adds an application to the application list.

Parameters:

command_socket Handle of command connection. Used to identify the application.

shmaddr_in Pointer to input (NIC to application) buffer in shared memory

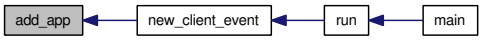
shmsize_in Size of input buffer

shmaddr_out Pointer to output (application to NIC) buffer in shared memory

shmsize_out Size of output buffer

Definition at line 61 of file register.c.

Here is the caller graph for this function:



16.9.2.2 filterid_t add_filter (sock_t *command_socket*, uint8_t * *filter_code*, int *filter_len*)

Adds a filter for a specific application (identified by it's socket).

Parameters:

command_socket Identifies the application by it's command connection handle

filter_code Pointer to the code block of the filter

filter_len Length of filter byte code

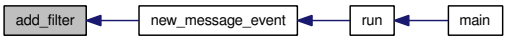
Returns:

Identifier of the filter (equal to it's index in the filter array)

Definition at line 110 of file register.c.

Here is the call graph for this function:

Here is the caller graph for this function:



16.9.2.3 int find_app_in_table (struct client_app ** *table*, int *table_len*, int *command_socket*)

Searches for the application's index given it's command socket handle.

Parameters:

table the application list to search in

table_len length of the list

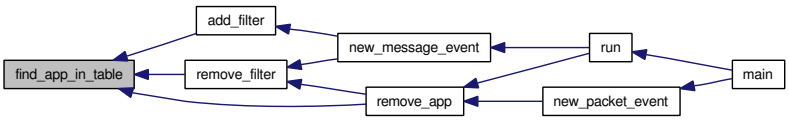
command_socket the command socket handle to look for

Returns:

the index of the application we found; -1 if none

Definition at line 296 of file register.c.

Here is the caller graph for this function:



16.9.2.4 void remove_app (sock_t *command_socket*)

Removes an application from the application list.

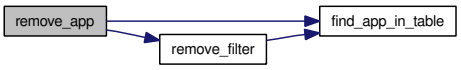
Removes any filters registered by this application and then removes the application from the list.

Parameters:

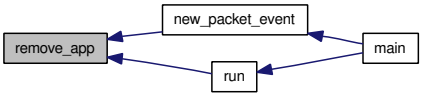
command_socket Identifies the application via it's command connection socket

Definition at line 171 of file register.c.

Here is the call graph for this function:



Here is the caller graph for this function:



16.9.2.5 void remove_filter (sock_t *command_socket*, filterid_t *filter_id*)

Removes a filter for an application.

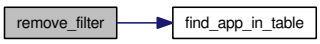
Parameters:

command_socket Identifies the application via it's command connection handle

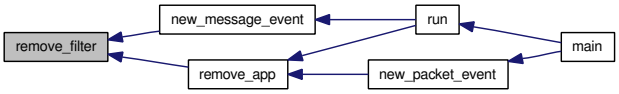
filter_id Identifier of the filter to be removed

Definition at line 231 of file register.c.

Here is the call graph for this function:



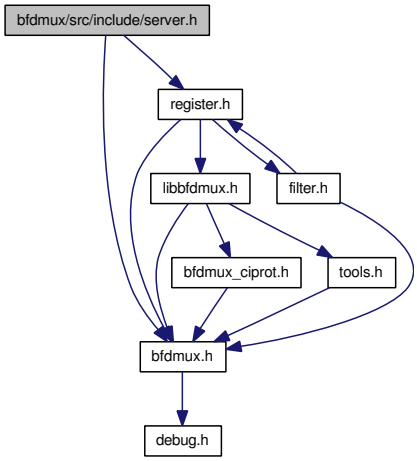
Here is the caller graph for this function:



16.10 bfdmux/src/include/server.h File Reference

Server thread header file.

Include dependency graph for server.h:



Defines

- #define MAX_PENDING 10

  Maximum number of pending, non-accepted connections on the listen socket.

- #define MAX_COMMAND_BUFFER_SIZE 100000

  Limits the size of a command packet, e.g. including a filter string.

- #define TIMEOUT_SEC 100

  Timeout for select command in the server loop.

Functions

- sock_t initialize_server ()

  Initializes and opens the servers listen socket.

- err_t close_server (pthread_t server_thread, sock_t server_socket)

  Closes the servers listen socket.

- void * run (void *socket)

  Main server function.

- err_t forward_packet_to_client (uint8_t *packet_data, int packet_len, struct client_app *receipient, filterid_t fid)

Forwards a packet to a single client.

16.10.1 Detailed Description

Server thread header file.

Definition in file server.h.

16.10.2 Function Documentation

16.10.2.1 err_t close_server (pthread_t *server_thread*, sock_t *server_socket*)

Closes the servers listen socket.

This closes command connections to all applications, detaches shared memory buffers and destroys the application register.

Parameters:

server_thread Server thread that has to be killed

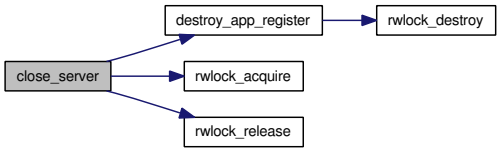
server_socket The handle to the server socket

Returns:

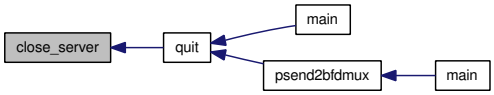
ERR_OK on success

Definition at line 233 of file server.c.

Here is the call graph for this function:



Here is the caller graph for this function:



16.10.2.2 err_t forward_packet_to_client (uint8_t * *packet_data*, int *packet_len*, struct client_app * *receipient*, filterid_t *fid*)

Forwards a packet to a single client.

Parameters:

packet_data Pointer to data to forward

packet_len Length of packet data in bytes

receipient Destination application that receives the data

fid Identifier of the filter that matched the packet

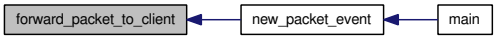
Returns:

ERR_OK on success, ERR_DISCONNECT if the caller should disconnect and remove the application, other error types on failure.

Definition at line 759 of file server.c.

Here is the call graph for this function:

Here is the caller graph for this function:



16.10.2.3 sock_t initialize_server ()

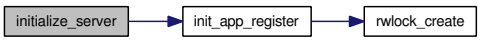
Initializes and opens the servers listen socket.

Returns:

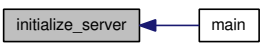
Returns the handle of the listen socket

Definition at line 148 of file server.c.

Here is the call graph for this function:



Here is the caller graph for this function:



16.10.2.4 void* run (void * *socket*)

Main server function.

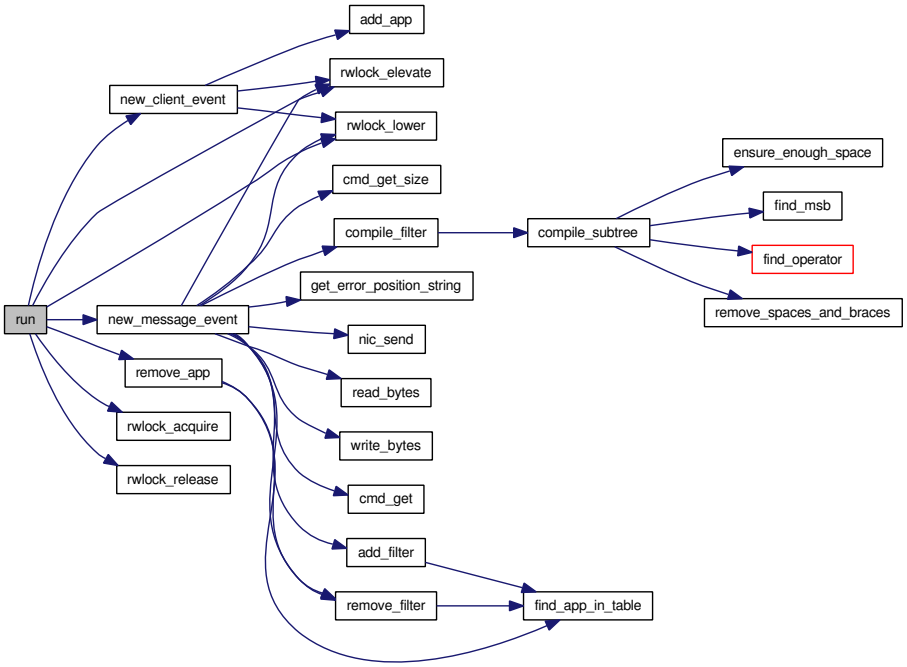
Contains the run loop of the server thread that waits for changes on the client command connections. If any changes on the socket descriptor set occurr, a handler function is called.

Parameters:

socket The listen socket

Definition at line 272 of file server.c.

Here is the call graph for this function:

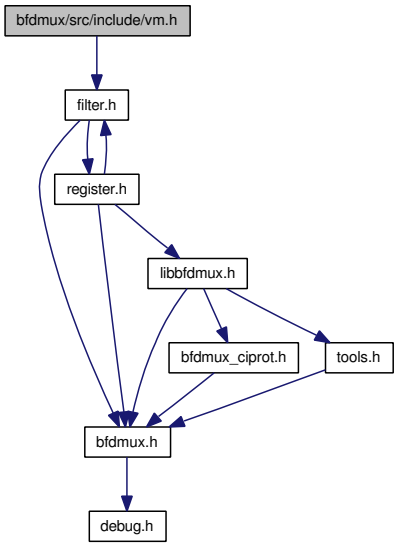


Here is the caller graph for this function:

16.11 bfdmux/src/include/vm.h File Reference

Interface for filter execution virtual machine.

Include dependency graph for vm.h:



Defines

- #define ERR_BAD_OP -1

  Execution failed because of an unknown opcode.

- #define ERR_BAD_ACCESS -2

  Filter did not match because it tried to access a non existing location in the packet.

- #define ERR_UNKNOWN -3

  An unknown internal error occurred during the execution.

Typedefs

- typedef uint8_t op_t

  Define opcode type as single byte.

Functions

- bool execute_filter (uint8_t ∗filter_code, int filter_len, uint8_t ∗packet_data, int packet_len, int ∗error_out)

Executes the specified filter on the given packet.

16.11.1 Detailed Description

Interface for filter execution virtual machine.

Definition in file vm.h.

16.11.2 Function Documentation

16.11.2.1 bool execute_filter (uint8_t * *filter_code*, int *filter_len*, uint8_t * *packet_data*, int *packet_len*, int * *error_out*)

Executes the specified filter on the given packet.

Parameters:

filter_code Points to the filters byte code

filter_len Length of the byte code

packet_data Points to the packet data to run the filter on

packet_len Length of packet data in bytes

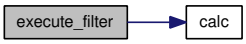
→ ***error_out*** Error information upon failure during execution

Returns:

true, if the filter executed successfully and the result was not zero. false otherwise.

Definition at line 344 of file vm.c.

Here is the call graph for this function:

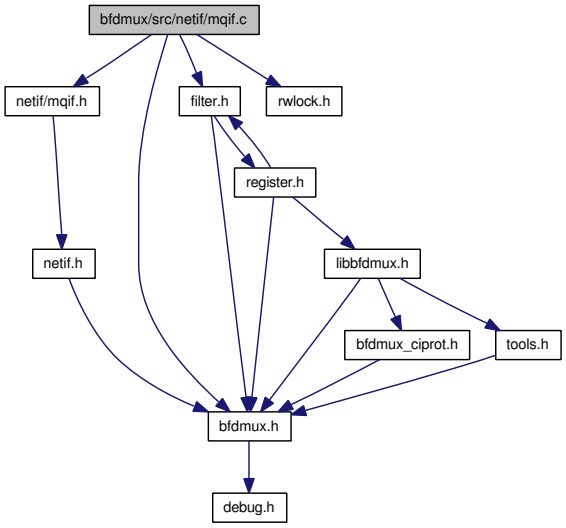


Here is the caller graph for this function:

16.12 bfdmux/src/netif/mqif.c File Reference

Sample network interface driver using two (in, out) message queues.

Include dependency graph for mqif.c:



Defines

- #define NUM_BUFS (PROC_QUEUE_LEN + 1)

  Define a certain number of buffers to allow a little queueing.

Functions

- void signal_handler (int signum)

  Signal handler that handles incoming packets (interrupted by a signal).

- err_t init_nic_interface ()

  Initialize the network card interface.

- err_t close_nic_interface ()

  Close the network driver.

- err_t nic_send (void *buff, size_t len)

  Send packets out to the world on the outbound message queue.

Variables

- struct sigaction sa

  Signal handler structure.

- mq_t mq_in

  Inbound (NIC to bfdmux) message queue ID.

- mq_t mq_out

  Outbound (bfdmux to NIC) message queue ID.

- struct nic_message_buf mbuf [NUM_BUFS]

  Message buffer array.

- int current_buf

  Buffer to use (pay attention: don't overwrite a buffer that is being processed!).

16.12.1 Detailed Description

Sample network interface driver using two (in, out) message queues.

This network interface driver is used for testing purposes. It comes with the msgq_-loopback project which is a simple loopback interface. This driver is implemented using message queues. When a packet is sent to this NIC, a signal a must be sent in addition to wake it up (just like real NICs with interrupts).

Definition in file mqif.c.

16.12.2 Define Documentation

16.12.2.1 #define NUM_BUFS (PROC_QUEUE_LEN + 1)

Define a certain number of buffers to allow a little queueing.

Note:

This should always be 1 larger than the request buffer 'PROC_QUEUE_LEN' of bfdmux, otherwise packets will be silently overridden during processing!

Definition at line 32 of file mqif.c.

16.12.3 Function Documentation

16.12.3.1 err_t close_nic_interface ()

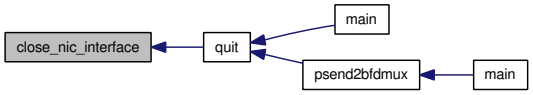
Close the network driver.

Returns:

ERR_OK on success

Definition at line 92 of file mqif.c.

Here is the caller graph for this function:



16.12.3.2 err_t init_nic_interface ()

Initialize the network card interface.

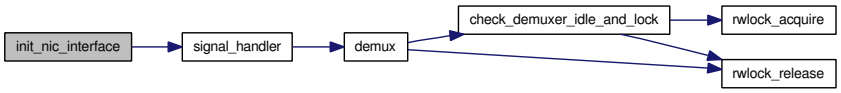
Set up signal handler and initialize the message queue to receive packets from the NIC.

Returns:

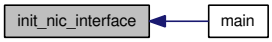
ERR_OK on success, ERR_FATAL on a fatal error.

Definition at line 51 of file mqif.c.

Here is the call graph for this function:



Here is the caller graph for this function:



16.12.3.3 err_t nic_send (void * *buff*, size_t *len*)

Send packets out to the world on the outbound message queue.

Parameters:

buff Pointer to the data array

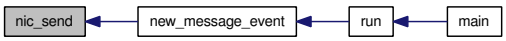
len Number of bytes to send starting at 'buff'

Returns:

ERR_OK on success, ERR_FATAL on error

Definition at line 144 of file mqif.c.

Here is the caller graph for this function:



16.12.3.4 void signal_handler (int *signum*)

Signal handler that handles incoming packets (interrupted by a signal).

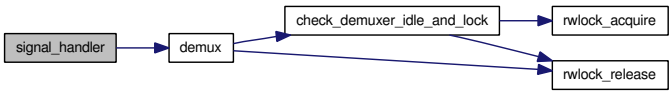
This is the signal/interrupt handler. After receiving data from the message queue the filter is invoked.

Parameters:

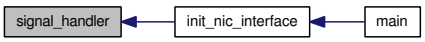
signum The signal number (should be SIGNICINTERRUPT)

Definition at line 108 of file mqif.c.

Here is the call graph for this function:



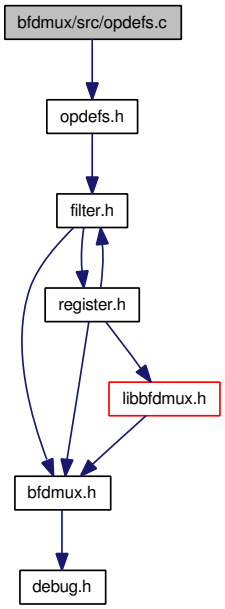
Here is the caller graph for this function:



16.13 bfdmux/src/opdefs.c File Reference

Bfdmux core functionality.

Include dependency graph for opdefs.c:



Variables

- op_def_t op_list [ ]
  List of operators and opcodes.

16.13.1 Detailed Description

Bfdmux core functionality.

Operator precedence definition and opcode/opstring binding

Definition in file opdefs.c.

16.13.2 Variable Documentation

16.13.2.1 op_def_t op_list[ ]

Initial value:

```
{
```

```
    {"||", OP_OR, 5, 0x11}
    ,
    {"&&", OP_AND, 5, 0x11}
    ,
    {"|", OP_BOR, 1, 0x11}
    ,
    {"^", OP_BXOR, 1, 0x11}
    ,
    {"&", OP_BAND, 1, 0x11}
    ,
    {"==", OP_EQUAL, 1, 0x11}
    ,
    {"!=", OP_UNEQUAL, 1, 0x11}
    ,
    {">=", OP_SGREATEREQUAL, 1, 0x11}
    ,
    {"<=", OP_SLESSEQUAL, 1, 0x11}
    ,
    {"}=", OP_UGREATEREQUAL, 1, 0x11}
    ,
    {"{=", OP_ULESSEQUAL, 1, 0x11}
    ,
    {">", OP_SGREATER, 1, 0x11}
    ,
    {"<", OP_SLESS, 1, 0x11}
    ,
    {"}", OP_UGREATER, 1, 0x11}
    ,
    {"{", OP_ULESS, 1, 0x11}
    ,
    {"+", OP_ADD, 1, 0x11}
    ,
    {"-", OP_SUB, 1, 0x11}
    ,
    {"*", OP_MULT, 1, 0x11}
    ,
    {"/", OP_IDIV, 1, 0x11}
    ,
    {"%", OP_MOD, 1, 0x11}
    ,
    {"!", OP_NOT, 1, 0x01}
    ,
    {"~", OP_BNOT, 1, 0x01}
    ,
    {"int8", OP_LOAD8, 1, 0x01}
    ,
    {"int16", OP_LOAD16, 1, 0x01}
    ,
    {"int32", OP_LOAD32, 1, 0x01}
    ,
    {"int64", OP_LOAD64, 1, 0x01}
    ,
    {"", 0, 0, 0}
}
```

List of operators and opcodes.

Operators with lower indices have lower precedence.

Warning:

If one operator string is contained in another one, the longer opstring needs to

reside in a lower index in the array! Example: "<" and "<=". If this is not the case the string "4 <= 5" will be split into "4" < "= 5", which will lead to a compile error in the right subexpression!

Note:

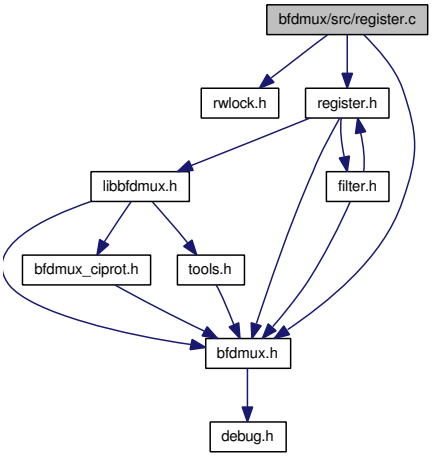
For OP_OR and OP_AND
5 specifies that 4 extra bytes should be left after the opcode: they will hold the left subtree length for skipping the right subtree evaluation if possible.
The empty operator signals the end of the operator list, do not remove it!

Definition at line 21 of file opdefs.c.

16.14 bfdmux/src/register.c File Reference

Application registration API.

Include dependency graph for register.c:



Functions

- void init_app_register (void)

  Initializes the application register app_table.

- void destroy_app_register (void)

  Cleans up application register app_table.

- void add_app (sock_t command_socket, void *shmaddr_in, size_t shmsize_in, void *shmaddr_out, size_t shmsize_out)

  Adds an application to the application list.

- filterid_t add_filter (sock_t command_socket, uint8_t *filter_code, int filter_len)

  Adds a filter for a specific application (identified by it's socket).

- void remove_app (sock_t command_socket)

  Removes an application from the application list.

- void remove_filter (sock_t command_socket, filterid_t filter_id)

  Removes a filter for an application.

- int find_app_in_table (struct client_app **table, int table_len, int command_socket)

  Searches for the application's index given it's command socket handle.

Variables

- struct client_app ** app_table

  Global table of connected client applications.

- int num_apps

  Length of app_table.

- int app_table_lock

  An rwlock handle that should be acquired before reading from or writing to the application list.

16.14.1 Detailed Description

Application registration API.

This file contains the implementation of the bfdmux application register.

Definition in file register.c.

16.14.2 Function Documentation

16.14.2.1 void add_app (sock_t *command_socket*, void * *shmaddr_in*, size_t *shmsize_in*, void * *shmaddr_out*, size_t *shmsize_out*)

Adds an application to the application list.

Parameters:

command_socket Handle of command connection. Used to identify the application.

shmaddr_in Pointer to input (NIC to application) buffer in shared memory

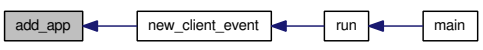
shmsize_in Size of input buffer

shmaddr_out Pointer to output (application to NIC) buffer in shared memory

shmsize_out Size of output buffer

Definition at line 61 of file register.c.

Here is the caller graph for this function:



16.14.2.2 filterid_t add_filter (sock_t *command_socket*, uint8_t * *filter_code*, int *filter_len*)

Adds a filter for a specific application (identified by it's socket).

Parameters:

command_socket Identifies the application by it's command connection handle
filter_code Pointer to the code block of the filter
filter_len Length of filter byte code

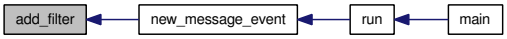
Returns:

Identifier of the filter (equal to it's index in the filter array)

Definition at line 110 of file register.c.

Here is the call graph for this function:

Here is the caller graph for this function:



16.14.2.3 int find_app_in_table (struct client_app ** *table*, int *table_len*, int *command_socket*)

Searches for the application's index given it's command socket handle.

Parameters:

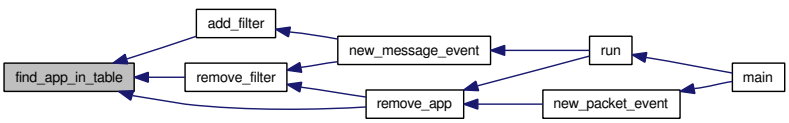
table the application list to search in
table_len length of the list
command_socket the command socket handle to look for

Returns:

the index of the application we found; -1 if none

Definition at line 296 of file register.c.

Here is the caller graph for this function:



16.14.2.4 void remove_app (sock_t *command_socket*)

Removes an application from the application list.

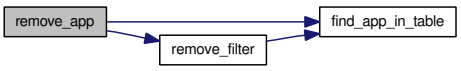
Removes any filters registered by this application and then removes the application from the list.

Parameters:

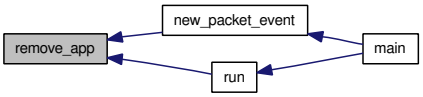
command_socket Identifies the application via it's command connection socket

Definition at line 171 of file register.c.

Here is the call graph for this function:



Here is the caller graph for this function:



16.14.2.5 void remove_filter (sock_t *command_socket*, filterid_t *filter_id*)

Removes a filter for an application.

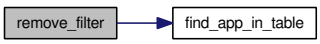
Parameters:

command_socket Identifies the application via it's command connection handle

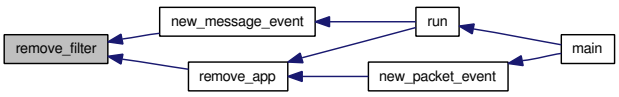
filter_id Identifier of the filter to be removed

Definition at line 231 of file register.c.

Here is the call graph for this function:



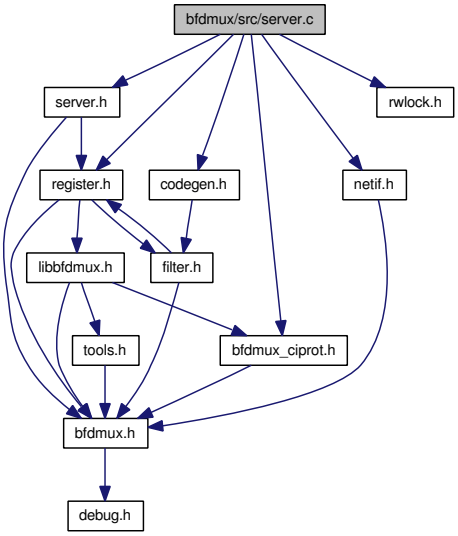
Here is the caller graph for this function:



16.15 bfdmux/src/server.c File Reference

Client application interface.

Include dependency graph for server.c:



Functions

- err_t new_client_event (sock_t server_socket)
  Event handler for newly connected application.
- err_t new_message_event (int app_id)
  Event handler for new data on an applications socket.
- void sighandler (int signum)
  Generic signal handler. Doesn't do anything at the moment.
- bool read_bytes (sock_t socket, void ∗buf, int length)
  Reads bytes from a given socket.
- bool write_bytes (sock_t socket, void ∗buf, int length)
  Writes bytes to the given socket.
- sock_t initialize_server ()
  Initializes and opens the servers listen socket.
- err_t close_server (pthread_t server_thread, sock_t server_socket)
  Closes the servers listen socket.

- void * run (void *socket)

  Main server function.

- err_t forward_packet_to_client (uint8_t *packet_data, int packet_len, struct client_app *receipient, filterid_t fid)

  Forwards a packet to a single client.

Variables

- struct sigaction sigact

  Signal handler for broken pipe signal.

16.15.1 Detailed Description

Client application interface.

This file implements the server that provides the socket connection for client applications.

Definition in file server.c.

16.15.2 Function Documentation

16.15.2.1 err_t close_server (pthread_t *server_thread*, sock_t *server_socket*)

Closes the servers listen socket.

This closes command connections to all applications, detaches shared memory buffers and destroys the application register.

Parameters:

server_thread Server thread that has to be killed

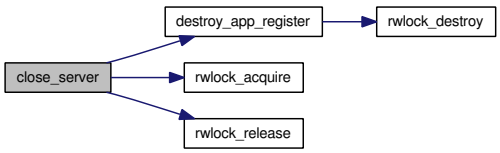
server_socket The handle to the server socket

Returns:

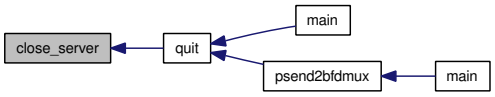
ERR_OK on success

Definition at line 233 of file server.c.

Here is the call graph for this function:



Here is the caller graph for this function:



16.15.2.2 err_t forward_packet_to_client (uint8_t * *packet_data*, int *packet_len*, struct client_app * *receipient*, filterid_t *fid*)

Forwards a packet to a single client.

Parameters:

packet_data Pointer to data to forward

packet_len Length of packet data in bytes

receipient Destination application that receives the data

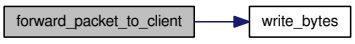
fid Identifier of the filter that matched the packet

Returns:

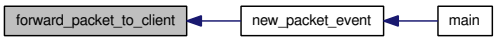
ERR_OK on success, ERR_DISCONNECT if the caller should disconnect and remove the application, other error types on failure.

Definition at line 759 of file server.c.

Here is the call graph for this function:



Here is the caller graph for this function:



16.15.2.3 sock_t initialize_server ()

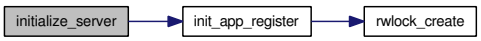
Initializes and opens the servers listen socket.

Returns:

Returns the handle of the listen socket

Definition at line 148 of file server.c.

Here is the call graph for this function:



Here is the caller graph for this function:

16.15.2.4 err_t new_client_event (sock_t *server_socket*)

Event handler for newly connected application.

Invoked by the servers main loop if a new connection from a client application has been established.

Warning:

The function assumes to have a valid read lock on the application table

Parameters:

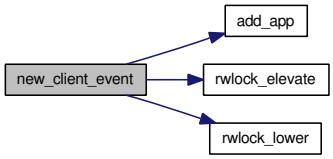
server_socket The server socket that received the new connection

Returns:

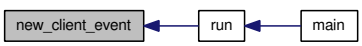
ERR_OK on success, other error types on failure

Definition at line 374 of file server.c.

Here is the call graph for this function:



Here is the caller graph for this function:



16.15.2.5 err_t new_message_event (int *app_id*)

Event handler for new data on an applications socket.

Parameters:

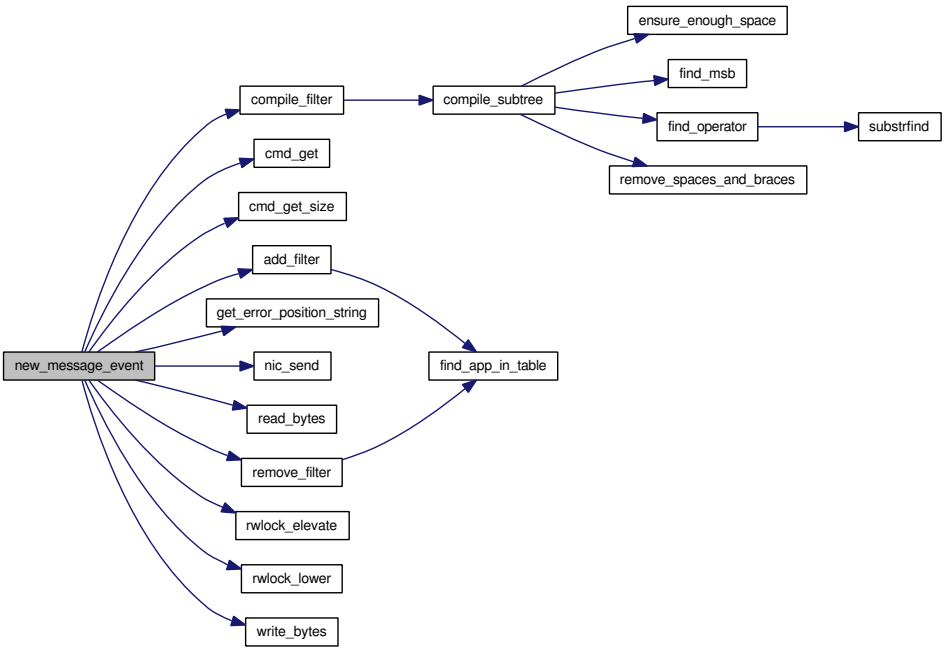
app_id Index of the application in the table from which we received a new command

Returns:

ERR_OK on success, ERR_DISCONNECT if the application should be disconnected and removed by the caller; other error types on failure.

Definition at line 408 of file server.c.

Here is the call graph for this function:



Here is the caller graph for this function:

16.15.2.6 bool read_bytes (sock_t *socket*, void * *buf*, int *length*)

Reads bytes from a given socket.

Parameters:

socket The socket handle to read from

buf The buffer to write to, if data is received

length The number of bytes to read. Will block until this number of bytes have arrived.

Returns:

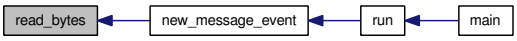
true, if the specified number of bytes could be read, otherwise false.

Note:

Specifying zero as length will return true and return immediately.

Definition at line 76 of file server.c.

Here is the caller graph for this function:



16.15.2.7 void* run (void * *socket*)

Main server function.

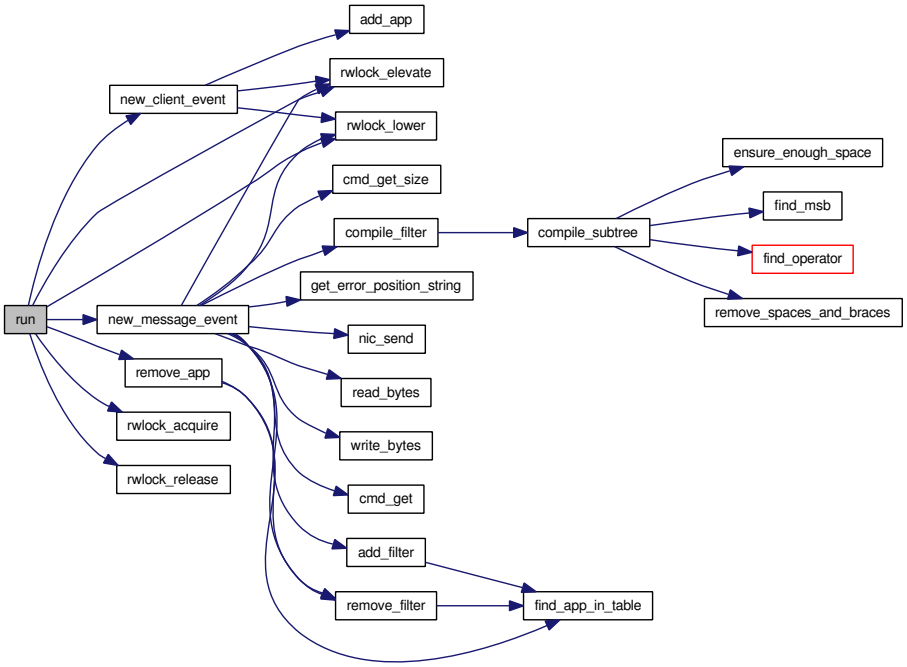
Contains the run loop of the server thread that waits for changes on the client command connections. If any changes on the socket descriptor set occurr, a handler function is called.

Parameters:

socket The listen socket

Definition at line 272 of file server.c.

Here is the call graph for this function:



Here is the caller graph for this function:

16.15.2.8 void sighandler (int *signum*)

Generic signal handler. Doesn't do anything at the moment.

Parameters:

signum The number of the signal for which the handler was invoked

Definition at line 60 of file server.c.

16.15.2.9 bool write_bytes (sock_t *socket*, void * *buf*, int *length*)

Writes bytes to the given socket.

Parameters:

socket The connection to write to

buf Pointer to the data

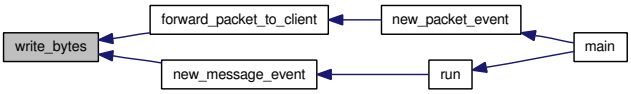
length The number of bytes to write

Returns:

true, if the specified number of bytes could be written, false otherwise.

Definition at line 110 of file server.c.

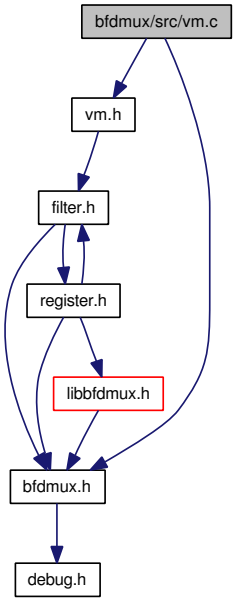
Here is the caller graph for this function:



16.16 bfdmux/src/vm.c File Reference

Implements a virtual machine for executing compiled intermediate language byte code.

Include dependency graph for vm.c:



Functions

- err_t calc (uint8_t *filter_code, int filter_len, uint8_t *packet_data, int packet_-len, uint64_t *result_value, size_t *result_offset)

  Performs recursive execution of a subtree of the filter code.

- bool execute_filter (uint8_t *filter_code, int filter_len, uint8_t *packet_data, int packet_len, int *error_out)

  Executes the specified filter on the given packet.

16.16.1 Detailed Description

Implements a virtual machine for executing compiled intermediate language byte code.

Definition in file vm.c.

16.16.2 Function Documentation

16.16.2.1 err_t calc (uint8_t * *filter_code*, int *filter_len*, uint8_t * *packet_data*, int *packet_len*, uint64_t * *result_value*, size_t * *result_offset*)

Performs recursive execution of a subtree of the filter code.

Parameters:

filter_code Points to the begining of the filter code

filter_len Specifies the length of the filter code in bytes

packet_data Points to the packet data to run the filter on

packet_len Specifies the length of the packet data in bytes

→ ***result_value*** Return value of the subtree execution

← ***result_offset*** Initially specifies the offset of the next byte to be executed in the filter code

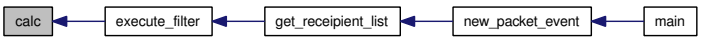
→ ***result_offset*** Specifies the next code byte to be executed, after the entire subtree code was executed

Returns:

ERR_OK on success, other error values on failure; see header file for error types.

Definition at line 37 of file vm.c.

Here is the caller graph for this function:



16.16.2.2 bool execute_filter (uint8_t * *filter_code*, int *filter_len*, uint8_t * *packet_data*, int *packet_len*, int * *error_out*)

Executes the specified filter on the given packet.

Parameters:

filter_code Points to the filters byte code

filter_len Length of the byte code

packet_data Points to the packet data to run the filter on

packet_len Length of packet data in bytes

→ ***error_out*** Error information upon failure during execution

Returns:

true, if the filter executed successfully and the result was not zero. false otherwise.

Definition at line 344 of file vm.c.

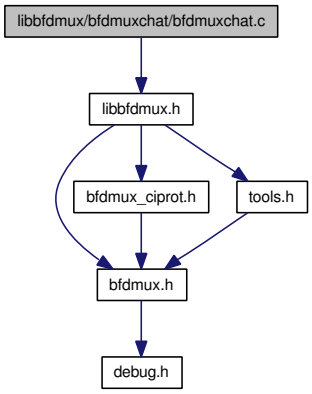
Here is the call graph for this function:

Here is the caller graph for this function:

16.17 libbfdmux/bfdmuxchat/bfdmuxchat.c File Reference

Sample chat application.

Include dependency graph for bfdmuxchat.c:



Functions

- void new_msg (void *msg, size_t len, filterid_t id)

  Event handler that gets called upon incoming packets.

- void quit (int signum)

  Destructor/Signal handler for CTRL+C keystork.

- int main ()

  Main routine.

16.17.1 Detailed Description

Sample chat application.

This is is sample application using the libbfdmux library. Launch multiple instances to chat over bfdmux.

Definition in file bfdmuxchat.c.

16.17.2 Function Documentation

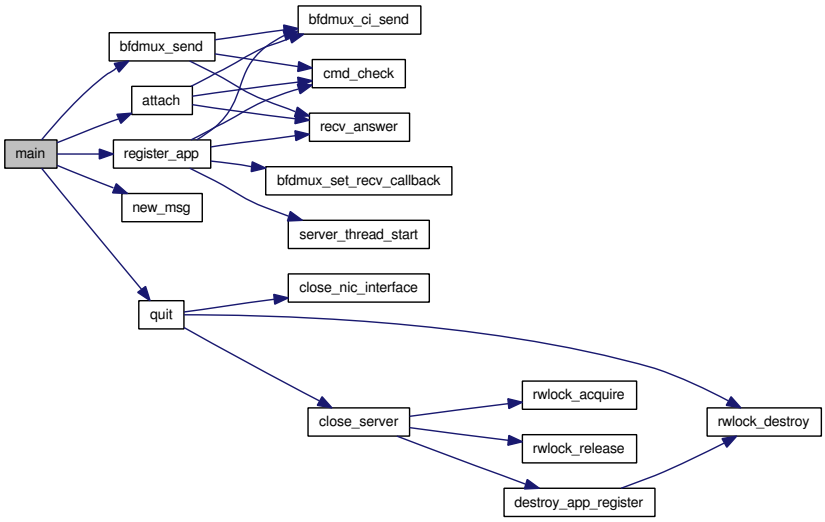
16.17.2.1 int main ()

Main routine.

This function registers the application at the bfdmux instance, asks the user for a nickname and builds the filter upon it. The filter will match all packets where the first byte is different to the first byte in the nickname. The filter gets attached and we jump into an endless chat loop.

Definition at line 79 of file bfdmuxchat.c.

Here is the call graph for this function:



16.17.2.2 void new_msg (void * *msg*, size_t *len*, filterid_t *id*)

Event handler that gets called upon incoming packets.

Parameters:

msg Pointer to chat message

len Message length

id Filter ID that matched for this packet

Definition at line 42 of file bfdmuxchat.c.

Here is the caller graph for this function:

16.17.2.3 void quit (int *signum*)

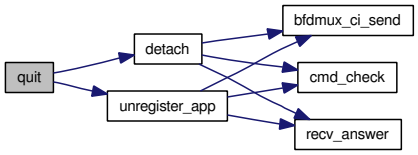
Destructor/Signal handler for CTRL+C keystork.

Parameters:

signum Signal that fired this signal handler (should be SIGINT)

Definition at line 57 of file bfdmuxchat.c.

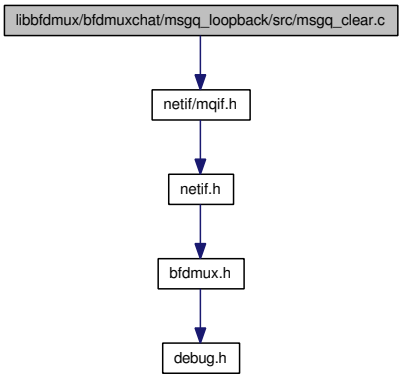
Here is the call graph for this function:



16.18 libbfdmux/bfdmuxchat/msgq_-loopback/src/msgq_clear.c File Reference

Clean all message queues.

Include dependency graph for msgq_clear.c:



16.18.1 Detailed Description

Clean all message queues.

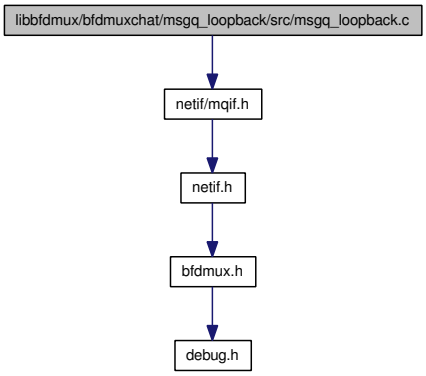
This application cleans all message queues that get used by the message queue interface from bfdmux.

Definition in file msgq_clear.c.

16.19 libbfdmux/bfdmuxchat/msgq_-loopback/src/msgq_loopback.c File Reference

Message queue loopback.

Include dependency graph for msgq_loopback.c:



16.19.1 Detailed Description

Message queue loopback.

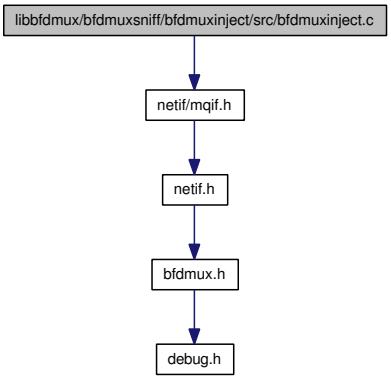
Outgoing packets from the message queue interface of bfdmux will be immediately re-injected.

Definition in file msgq_loopback.c.

16.20 libbfdmux/bfdmuxsniff/bfdmuxinject/src/bfdmuxinject.c File Reference

Inject real network packets from your 'to-the-world-connected' NIC into bfdmux.

Include dependency graph for bfdmuxinject.c:



Functions

- void quit (int signum)

  Exit handler.

- void psend2bfdmux (u_char *data, size_t len)

  Forward packet to bfdmux.

- void pprint (u_char *data, size_t len)

  Print packet to console.

- int main (int argc, char **argv)

  Main function with endless loop for packet capturing.

16.20.1 Detailed Description

Inject real network packets from your 'to-the-world-connected' NIC into bfdmux.

This tool captures all packets on a given interface and forwards them to your bfdmux instance. Capturing is done using the pcap library.

Definition in file bfdmuxinject.c.

16.20.2 Function Documentation

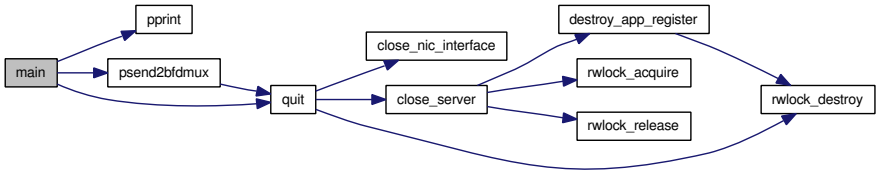
16.20.2.1 int main (int *argc*, char ** *argv*)

Main function with endless loop for packet capturing.

This function initializes a signal handler to exit the program, the message queue to communicate with bfdmux and pcap to capture packets. Packet capturing is done in an endless loop. To quit press CTRL+C.

Definition at line 103 of file bfdmuxinject.c.

Here is the call graph for this function:



16.20.2.2 void pprint (u_char * *data*, size_t *len*)

Print packet to console.

Parameters:

data Pointer to data segment

len Data length

Definition at line 83 of file bfdmuxinject.c.

Here is the caller graph for this function:

16.20.2.3 void psend2bfdmux (u_char * *data*, size_t *len*)

Forward packet to bfdmux.

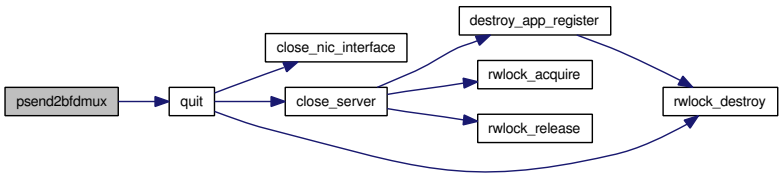
Parameters:

data Pointer to data segment

len Data length

Definition at line 50 of file bfdmuxinject.c.

Here is the call graph for this function:

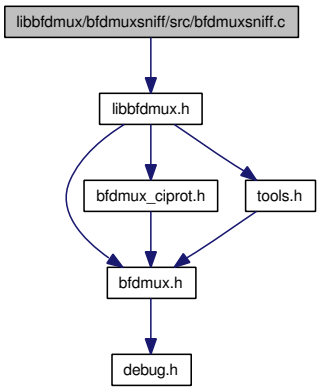


Here is the caller graph for this function:

16.21 libbfdmux/bfdmuxsniff/src/bfdmuxsniff.c File Reference

A sniffer written for bfdmux.

Include dependency graph for bfdmuxsniff.c:



Functions

- void printicmpinfo (uint8_t *msg, size_t len)
  This function extracts metadata from an ICMP packet.
- void printtcpinfo (uint8_t *msg, size_t len)
  This function extracts metadata from an TCP packet.
- void printudpinfo (uint8_t *msg, size_t len)
  This function extracts metadata from an UDP packet.
- void printipv4info (uint8_t *msg, size_t len)
  This function extracts metadata from an IPv4 packet.
- void new_msg (void *msg, size_t len, filterid_t id)
  This handler gets called when the application received a new packet.
- void get_new_filter (int signum)
  This signal handler gets called when you want to enter a new filter.
- void quit (int signum)
  This signal handler gets called when you want to quit the sniffer.
- int main (int argc, char **argv)
  The main function sets up the signal handler, registers itself at bfdmux and waits in an endless loop for your interaction.

16.21.1 Detailed Description

A sniffer written for bfdmux.

With this sniffer you have access to all packets coming in into bfdmux. Additionally you can specify a personal filter (by hitten CTRL+\). The output fill be metadata of the fetched packets.

Currently bfdmuxsniff supports tcp, udp, icmp

Definition in file bfdmuxsniff.c.

16.21.2 Function Documentation

16.21.2.1 void new_msg (void * *msg*, size_t *len*, filterid_t *id*)

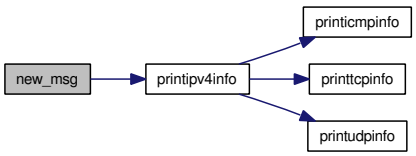
This handler gets called when the application received a new packet.

Parameters:

msg Pointer to packet data
len Packet length
id Filter ID that matched this packet

Definition at line 200 of file bfdmuxsniff.c.

Here is the call graph for this function:



16.21.2.2 void printicmpinfo (uint8_t * *msg*, size_t *len*)

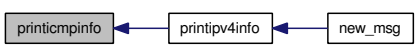
This function extracts metadata from an ICMP packet.

Parameters:

msg Pointer to ICMP packet
len Packet length

Definition at line 43 of file bfdmuxsniff.c.

Here is the caller graph for this function:



16.21.2.3 void printipv4info (uint8_t * *msg*, size_t *len*)

This function extracts metadata from an IPv4 packet.

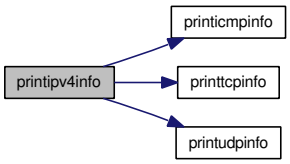
Parameters:

msg Pointer to ICMP packet

len Packet length

Definition at line 133 of file bfdmuxsniff.c.

Here is the call graph for this function:



Here is the caller graph for this function:

16.21.2.4 void printtcpinfo (uint8_t * *msg*, size_t *len*)

This function extracts metadata from an TCP packet.

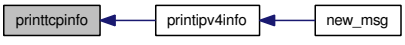
Parameters:

msg Pointer to ICMP packet

len Packet length

Definition at line 90 of file bfdmuxsniff.c.

Here is the caller graph for this function:



16.21.2.5 void printudpinfo (uint8_t * *msg*, size_t *len*)

This function extracts metadata from an UDP packet.

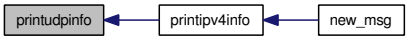
Parameters:

msg Pointer to ICMP packet

len Packet length

Definition at line 116 of file bfdmuxsniff.c.

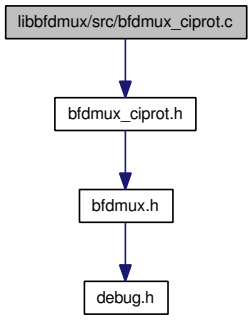
Here is the caller graph for this function:



16.22 libbfdmux/src/bfdmux_ciprot.c File Reference

Bfdmux client protocol interface implementation.

Include dependency graph for bfdmux_ciprot.c:



Functions

- cmd_t cmd_get (void *data)
  Get command type.
- size_t cmd_get_size (cmd_t cmd)
  Get command size.
- err_t cmd_check (cmd_t cmd, void *data, size_t len)
  Check if command is valid.

16.22.1 Detailed Description

Bfdmux client protocol interface implementation.

Helper functions for client command handling

Definition in file bfdmux_ciprot.c.

16.22.2 Function Documentation

16.22.2.1 err_t cmd_check (cmd_t *cmd*, void * *data*, size_t *len*)

Check if command is valid.

Verifies validity of a command packet using the command type and its size as reference.

Note:

Not very nice implementation

Parameters:

cmd Command type to check against

data Pointer to command packet data

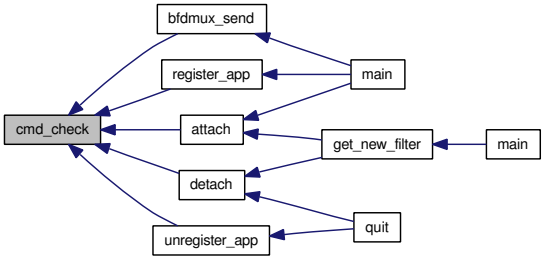
len Length of the packet

Returns:

If the command packet at 'data' is of type 'cmd' and has length 'len', this function returns CMD_OK. CMD_ERR if this is not the case.

Definition at line 96 of file bfdmux_ciprot.c.

Here is the caller graph for this function:



16.22.2.2 cmd_t cmd_get (void * *data*)

Get command type.

Extracts the command type to a given data string.

Parameters:

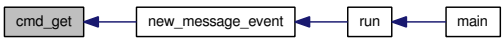
data A pointer to a data string containing a command packet.

Returns:

The command type upon success or CMD_ERR if the packet was not recognized.

Definition at line 23 of file bfdmux_ciprot.c.

Here is the caller graph for this function:



16.22.2.3 size_t cmd_get_size (cmd_t *cmd*)

Get command size.

Parameters:

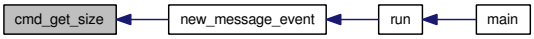
cmd Command type

Returns:

Size of an accurate command packet of type 'cmd'

Definition at line 53 of file bfdmux_ciprot.c.

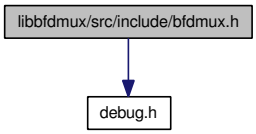
Here is the caller graph for this function:



16.23 libbfdmux/src/include/bfdmux.h File Reference

Bfdmux twek options.

Include dependency graph for bfdmux.h:



Defines

- #define PROC_QUEUE_LEN 5

  Number of NIC buffers to queue for processing.

- #define BFDMUX_SOCK_PATH "/tmp/.bfdmux.sock"

  Location of the UNIX socket file.

- #define FLUSH_AND_SYNC 1

  Always call fflush and sync on file descriptor and communication channels.

- #define ERR_OK 0

  No error.

- #define ERR_NONFATAL -1

  Error, but nonfatal.

- #define ERR_FATAL -2

  Fatal error, shut down.

- #define ERR_DISCONNECT -3

  Disconnect error. Client-server connection is lost.

- #define ERR_DROPPED -4

  Application was not able to receive a new packet. So the packet was dropped.

- #define PROTO_TCP 0x06

  TCP protocol number in IPv4 header.

- #define PROTO_UDP 0x11a

  UDP protocol number in IPv4 header.

- #define PORT_ANY 0x00

  Any UDP/TCP port.

- #define IP_ADDR_ANY 0x00
  Any IPv4-Address.
- #define IP_ADDR_LOCAL 0x7f000001
  This is the localhost 127.0.0.1 IP-Address.

Typedefs

- typedef uint8_t prot_t
  Protocol type.
- typedef uint32_t addr_t
  IP-Address type.
- typedef uint16_t port_t
  Port type.
- typedef int8_t err_t
  Error type.
- typedef int32_t sock_t
  Socket type.
- typedef int32_t mq_t
  Message queue type.
- typedef uint8_t cmd_t
  Command type.
- typedef uint32_t mqkey_t
  Message queue key type.
- typedef uint32_t smkey_t
  Shared memory key type.
- typedef int32_t filterid_t
  Filter id type. Negative values for errors.

Functions

- bool demux (void *data, int len)
  Tries to demux the given packet and forward it to the application.

16.23.1 Detailed Description

Bfdmux twek options.

Definition in file bfdmux.h.

16.23.2 Function Documentation

16.23.2.1 bool demux (void * *data*, int *len*)

Tries to demux the given packet and forward it to the application.

Parameters:

data Points to the packet in memory

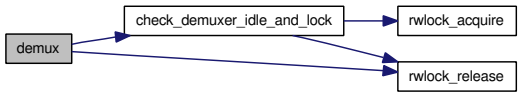
len Length of the packet data in bytes

Returns:

true if the packet could be put into the demuxer's queue; otherwise false.

Definition at line 133 of file bfdmux.c.

Here is the call graph for this function:

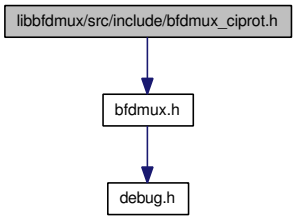


Here is the caller graph for this function:

16.24 libbfdmux/src/include/bfdmux_ciprot.h File Reference

Bfdmux command interface protocol header file Declaration of the available command packets and the corresponding command types.

Include dependency graph for bfdmux_ciprot.h:



Data Structures

- struct cmd_register
  Register command.
- struct cmd_register_answer
  Answer command to the register command.
- struct cmd_unregister
  Unregister command.
- struct cmd_unregister_answer
  Answer command to the unregister command.
- struct cmd_attach
  Attach command Attach a filter to the application.
- struct cmd_attach_answer
  Answer command to the attach command.
- struct cmd_detach
  Detach command.
- struct cmd_detach_answer
  Answer command to the detach command.
- struct cmd_send
  Send command.
- struct cmd_send_answer

Answer command to the send command.

- struct cmd_recv

  Receive command.

- struct cmd_recv_answer

  Answer command to the receive command.

- struct cmd_error

  Error command.

Defines

- #define CMD_ERROR 0xFF

  Error command.

- #define CMD_REGISTER 0xC0

  ID of the register command.

- #define CMD_UNREGISTER 0xC1

  ID of the unregister command.

- #define CMD_ATTACH 0xC2

  ID of the command to attach the application to a filter.

- #define CMD_DETACH 0xC3

  ID of the command to detach the application from a filter.

- #define CMD_SEND 0xC4

  ID of the command to send data out to the world.

- #define CMD_GET_IP_LIST 0xC6

  ID of the command to ask bfdmux for available IP-Addresses.

- #define CMD_RECV_ANSWER 0xA5

  ID of command to notice bfdmux that the application finished processing a data packet.

- #define CMD_REGISTER_ANSWER 0xA0

  ID of answer to the register command.

- #define CMD_UNREGISTER_ANSWER 0xA1

  ID of answer to the unregister command.

- #define CMD_ATTACH_ANSWER 0xA2

ID of answer to the attach command.

- #define CMD_DETACH_ANSWER 0xA3
  ID of answer to the detach command.
- #define CMD_SEND_ANSWER 0xA4
  ID of answer to the send command.
- #define CMD_GET_IP_LIST_ANSWER 0xA6
  ID of answer to the get IP-Address list command.
- #define CMD_RECV 0xC5
- #define CMD_OK 0
  Command is OK.
- #define CMD_ERR 0xff
  Error.

Functions

- cmd_t cmd_get (void *data)
  Get command type.
- size_t cmd_get_size (cmd_t cmd)
  Get command size.
- err_t cmd_check (cmd_t cmd, void *data, size_t len)
  Check if command is valid.

16.24.1 Detailed Description

Bfdmux command interface protocol header file Declaration of the available command packets and the corresponding command types.

Definition in file bfdmux_ciprot.h.

16.24.2 Define Documentation

16.24.2.1 #define CMD_GET_IP_LIST 0xC6

ID of the command to ask bfdmux for available IP-Addresses.

Warning:

Not implemented

Definition at line 24 of file bfdmux_ciprot.h.

16.24.2.2 #define CMD_GET_IP_LIST_ANSWER 0xA6

ID of answer to the get IP-Address list command.

Warning:

Not implemented

Definition at line 36 of file bfdmux_ciprot.h.

16.24.2.3 #define CMD_RECV 0xC5

ID of the command to notice the application that a new packet arrived

Definition at line 40 of file bfdmux_ciprot.h.

16.24.3 Function Documentation

16.24.3.1 err_t cmd_check (cmd_t *cmd*, void * *data*, size_t *len*)

Check if command is valid.

Verifies validity of a command packet using the command type and its size as reference.

Note:

Not very nice implementation

Parameters:

cmd Command type to check against

data Pointer to command packet data

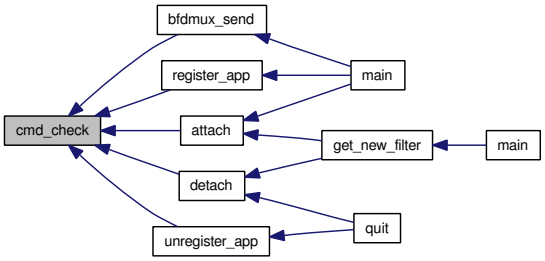
len Length of the packet

Returns:

If the command packet at 'data' is of type 'cmd' and has length 'len', this function returns CMD_OK. CMD_ERR if this is not the case.

Definition at line 96 of file bfdmux_ciprot.c.

Here is the caller graph for this function:



16.24.3.2 cmd_t cmd_get (void * *data*)

Get command type.

Extracts the command type to a given data string.

Parameters:

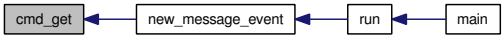
data A pointer to a data string containing a command packet.

Returns:

The command type upon success or CMD_ERR if the packet was not recognized.

Definition at line 23 of file bfdmux_ciprot.c.

Here is the caller graph for this function:



16.24.3.3 size_t cmd_get_size (cmd_t *cmd*)

Get command size.

Parameters:

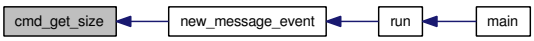
cmd Command type

Returns:

Size of an accurate command packet of type 'cmd'

Definition at line 53 of file bfdmux_ciprot.c.

Here is the caller graph for this function:



16.25 libbfdmux/src/include/debug.h File Reference

Debug makro definitions.

Defines

- #define DEBUG_LEVEL 2

  Debug level.

- #define PDEBUG_FNAME(x) char* __DEBUG__CURRENT_FUNCTION_NAME = x; int __DEBUG__OMIT = 0; if (__DEBUG__OMIT) {};

  Set the current function name for well-arranged debug messages.

- #define PDEBUG_OMIT __DEBUG__OMIT = 1;

  When calling this makro, all debug messages for the caller function will be omitted.

- #define PDEBUG_HEADER(x)

  *This prints a debug header e.g: libbfdmux.c:regster_app: *** Hello world debug message ***.*

- #define PDEBUG_FOOTER(x)

  This prints a debug footer line e.g: libbfdmux.c:regiser_app ### Foo bar footer ###.

- #define PDEBUG_ERROR(x)

  This makro is used to print error messages.

- #define PDEBUG_INFO(x)

  This makro is used to print additional information.

- #define PDEBUG_RAW(arr, cnt)

  This makro is used to dump a memory segment to the screen as hex and.

16.25.1 Detailed Description

Debug makro definitions.

Definition in file debug.h.

16.25.2 Define Documentation

16.25.2.1 #define DEBUG_LEVEL 2

Debug level.

- 0: No messages will be printed on stdout.

- 1: Only error messages will be printed.
- 2: Error and information messages will be printed.
- 3: Information, errors and packet data as ascii will be printed.
- 4: Information, errors and packet data as ascii and hex will be printed.

Definition at line 20 of file debug.h.

16.25.2.2 #define PDEBUG_RAW(arr, cnt)

Value:

```
if ((DEBUG_LEVEL >= 4) && (!__DEBUG__OMIT)) { \
                                    int __debug_i; __debug_i = 0; \
                                    char __debug_c; __debug_c = ' '; \
                                    printf("  %s: ", __DEBUG__CURRENT_FUNCTION_NAME); \
                                    printf("Address: %p, size: %u Bytes\n", arr, (unsigned) (cnt)); \
                                    printf("  %s: ", __DEBUG__CURRENT_FUNCTION_NAME); \
                                    for(__debug_i = 0; __debug_i < (cnt); __debug_i++) { \
                                        printf("%02x ",*( (uint8_t*) (arr) + __debug_i )); \
                                        if (!((__debug_i+1)%20)) printf("\n  %s: ", __DEBUG__CURRENT_FU
                                    }; \
                                }; \
                                if ((DEBUG_LEVEL >= 3) && (!__DEBUG__OMIT)) { \
                                    int __debug_j; __debug_j = 0; \
                                    char __debug_d; __debug_d = ' '; \
                                    printf("\n  %s: ", __DEBUG__CURRENT_FUNCTION_NAME); \
                                    for(__debug_j = 0; __debug_j < (cnt); __debug_j++) { \
                                        __debug_d = *((uint8_t*) (arr) + __debug_j ); \
                                        if (__debug_d < 0x20 || __debug_d > 0x7e) __debug_d = '*'; \
                                        printf("%c",__debug_d); \
                                        if (!((__debug_j+1)%60)) printf("\n  %s: ", __DEBUG__CURRENT_FU
                                    }; \
                                    printf("\n"); \
                                    if (FLUSH_AND_SYNC) { \
                                        fflush(stdin); \
                                    } \
                                };
```

This makro is used to dump a memory segment to the screen as hex and.

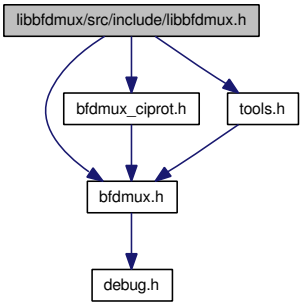
(if the debug level allows it) as characters.

Definition at line 102 of file debug.h.

16.26 libbfdmux/src/include/libbfdmux.h File Reference

Libbfdmux API.

Include dependency graph for libbfdmux.h:



Defines

- #define OK 0

  No error.

- #define ERR -1

  Error.

Functions

- err_t register_app (void(∗recv_callback)(void ∗, size_t, filterid_t), void ∗∗sm_inpt, void ∗∗sm_outpt, size_t size_in, size_t size_out)

  Register application at bfdmux.

- err_t unregister_app (void)

  Deregister application from bfdmux.

- filterid_t attach (char ∗filter)

  Attach application to a filter.

- err_t detach (filterid_t filter_id)

  Detach application from a previous attached filter with ID 'filter_id'.

- size_t bfdmux_send (size_t len)

  Send data out.

- void bfdmux_set_recv_callback (void(∗callback)(void ∗, size_t, filterid_t))

Set callback function for the receive packet event.

16.26.1 Detailed Description

Libbfdmux API.

Bfdmux interface (libbfdmux) for applications that want to use bfdmux.

Definition in file libbfdmux.h.

16.26.2 Function Documentation

16.26.2.1 filterid_t attach (char * *filter*)

Attach application to a filter.

Sends filter string 'filter' to bfdmux and attaches it.

Parameters:

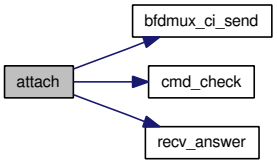
filter Pointer to the filter string

Returns:

Filter ID on success, -1 otherwise

Definition at line 326 of file libbfdmux.c.

Here is the call graph for this function:



Here is the caller graph for this function:

16.26.2.2 size_t bfdmux_send (size_t *len*)

Send data out.

Send 'len' bytes of data starting at the 'shmaddr_out' address out. The outbound buffer should be filled before calling this function.

Parameters:

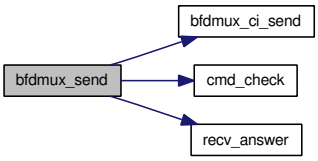
len Amount of bytes to send

Returns:

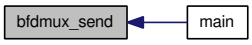
Number of bytes actually sent

Definition at line 436 of file libbfdmux.c.

Here is the call graph for this function:



Here is the caller graph for this function:



16.26.2.3 void bfdmux_set_recv_callback (void(∗)(void ∗, size_t, filterid_t) *callback*)

Set callback function for the receive packet event.

Parameters:

callback Function pointer to a receive-data handler

Definition at line 537 of file libbfdmux.c.

Here is the caller graph for this function:

16.26.2.4 err_t detach (filterid_t *filter_id*)

Detach application from a previous attached filter with ID 'filter_id'.

Parameters:

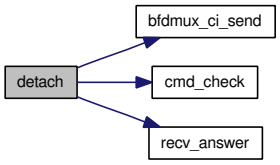
filter_id Filter ID of a previous attached filter

Returns:

OK on success, ERR otherwise.

Definition at line 384 of file libbfdmux.c.

Here is the call graph for this function:



Here is the caller graph for this function:

16.26.2.5 err_t register_app (void(∗)(void ∗, size_t, filterid_t) *recv_callback*, void ∗∗ *sm_inpt*, void ∗∗ *sm_outpt*, size_t *size_in*, size_t *size_out*)

Register application at bfdmux.

Starts thread to handle incoming commands, sets up shared memory of specified size, connects to and registers with bfdmux via Unix socket

Parameters:

recv_callback pointer to callback function for handling incoming data

→ ***sm_inpt*** pointer to a pointer to the incoming packet buffer (will be set on success)

→ ***sm_outpt*** pointer to a pointer to the outgoing packet buffer (will be set on success)

size_in desired size of inbound buffer

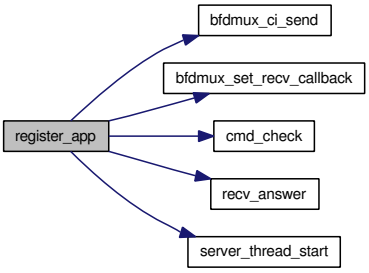
size_out desired size of outbound buffer

Returns:

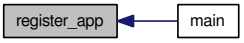
OK on success, otherwise ERR.

Definition at line 105 of file libbfdmux.c.

Here is the call graph for this function:



Here is the caller graph for this function:



16.26.2.6 err_t unregister_app (void)

Deregister application from bfdmux.

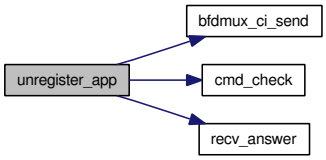
Deregisters from bfdmux, detaches shared memory and then closes command socket

Returns:

Returns OK if successful, otherwise ERR

Definition at line 252 of file libbfdmux.c.

Here is the call graph for this function:



Here is the caller graph for this function:

16.27 libbfdmux/src/include/rwlock.h File Reference

Read/write lock header file.

Defines

- #define MAX_READ_LOCKS 100

  Maximum number of simultaneous read-only locks on an object.

Functions

- int rwlock_create (void)

  Create a read/write lock handle.

- bool rwlock_destroy (int sid)

  Destroy a previous created read/write lock.

- bool rwlock_acquire (int sid, bool write)

  Acquire rights (read-write or read-only) on an existing lock.

- bool rwlock_elevate (int sid)

  Elevate a read-only lock to a read-write lock.

- bool rwlock_try_acquire (int sid, bool write)

  Try to acquire rights (read-write or read-only) on an existing lock.

- bool rwlock_release (int sid, bool write)

  Release a previous acquired right on a read/write lock.

- bool rwlock_lower (int sid)

  Lower the permission on a read/write lock.

16.27.1 Detailed Description

Read/write lock header file.

Definition in file rwlock.h.

16.27.2 Function Documentation

16.27.2.1 bool rwlock_acquire (int *sid*, bool *write*)

Acquire rights (read-write or read-only) on an existing lock.

Acquire a read-write or a read-only right on a previous created read/write lock. This function call is blocking and will return only after successfully acquiring the asked right or on error.

Parameters:

sid Read/write lock id

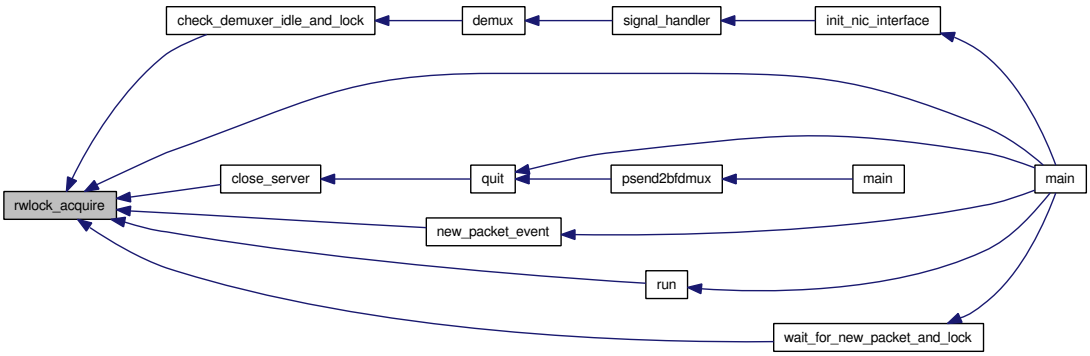
write Boolean argument to choose between a read-write lock (true) or a read-only lock (false)

Returns:

True on success, false otherwise.

Definition at line 79 of file rwlock.c.

Here is the caller graph for this function:



16.27.2.2 int rwlock_create (void)

Create a read/write lock handle.

Returns:

Read/write lock handle on success, -1 otherwise.

Definition at line 26 of file rwlock.c.

Here is the caller graph for this function:

16.27.2.3 bool rwlock_destroy (int *sid*)

Destroy a previous created read/write lock.

Parameters:

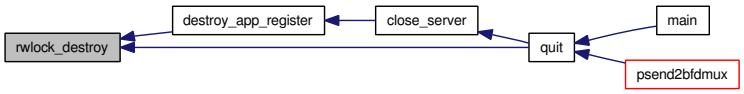
sid Previous created read/write lock that should get destroyed

Returns:

True on success, false otherwise.

Definition at line 61 of file rwlock.c.

Here is the caller graph for this function:



16.27.2.4 bool rwlock_elevate (int *sid*)

Elevate a read-only lock to a read-write lock.

Elevate the right on lock from read-only to read-write. This function is blocking and will return only after successfully elevating the read/write lock or after an error.

Parameters:

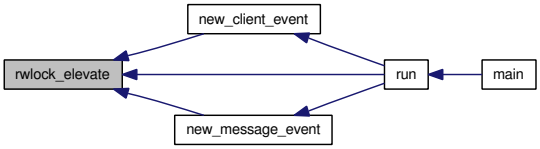
sid Read/write lock ID to elevate the rights on

Returns:

True on success, false otherwise.

Definition at line 126 of file rwlock.c.

Here is the caller graph for this function:



16.27.2.5 bool rwlock_lower (int *sid*)

Lower the permission on a read/write lock.

This function lowers the permission on a read/write lock from read-write to read-only. To completely remove the read/write lock use rwlock_release().

Parameters:

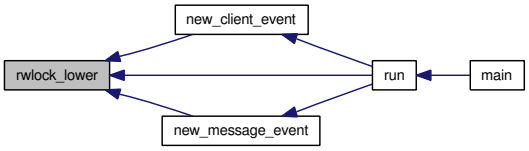
sid Read/write lock id

Returns:

True on success, false otherwise.

Definition at line 251 of file rwlock.c.

Here is the caller graph for this function:



16.27.2.6 bool rwlock_release (int *sid*, bool *write*)

Release a previous acquired right on a read/write lock.

Release a read/write lock. To lower the right from read-write to read-only don't use this function, use rwlock_lower(). This function completely releases the read/write lock.

Parameters:

sid Read/write lock id

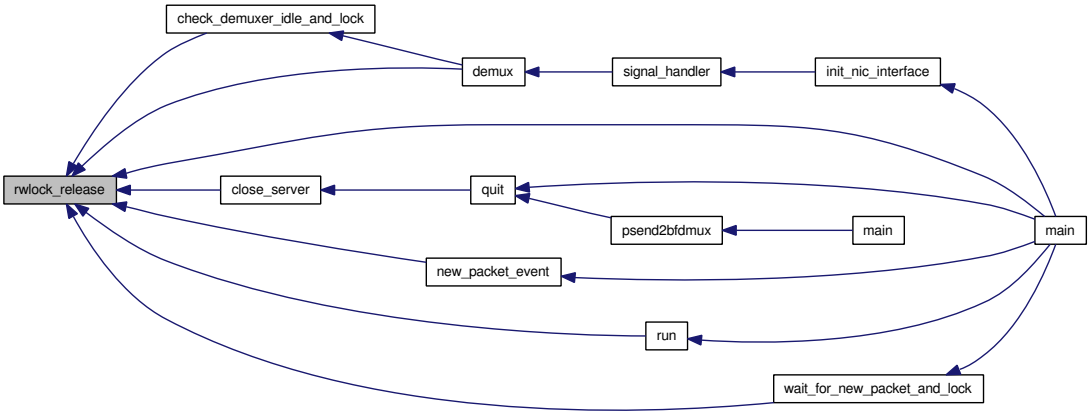
write Boolean argument specifying if this read/write lock has read-write (true) or read-only (false) permissions.

Returns:

True on success, false otherwise.

Definition at line 209 of file rwlock.c.

Here is the caller graph for this function:



16.27.2.7 bool rwlock_try_acquire (int *sid*, bool *write*)

Try to acquire rights (read-write or read-only) on an existing lock.

Acquire a read-write or a read-only right on a previous created read/write lock. This function call is non-blocking and will return immediately.

Parameters:

sid Read/write lock id

write Boolean argument to choose between a read-write lock (true) or a read-only lock (false)

Returns:

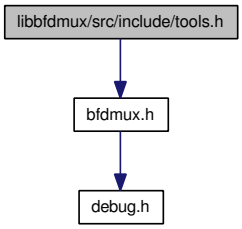
True on success, false otherwise.

Definition at line 167 of file rwlock.c.

16.28 libbfdmux/src/include/tools.h File Reference

Header file for helper and additional functions.

Include dependency graph for tools.h:



Functions

- int find_msb (uint64_t value)

  Finds the index of the most significant 1-bit in 'value'.

- uint8_t * parse_hex_input (char *str)

  Parses a string consisting of hex digits to a byte array.

- char * get_error_position_string (int pos)

  Returns a string with pos-1 spaces and a '^' character. Used to indicate error position in filter string!

- char * build_ipv4_filter (addr_t srcip, addr_t dstip)

  IPv4 filter template.

- char * build_tcp_filter (port_t srcport, port_t dstport)

  TCP filter template.

- char * build_udp_filter (port_t srcport, port_t dstport)

  UDP filter template.

- char * build_ipv4_tcp_filter (addr_t srcip, addr_t dstip, port_t srcport, port_t dstport)

  TCP over IPv4 filter template.

- char * build_ipv4_udp_filter (addr_t srcip, addr_t dstip, port_t srcport, port_t dstport)

  UDP over IPv4 filter template.

16.28.1 Detailed Description

Header file for helper and additional functions.

Definition in file tools.h.

16.28.2 Function Documentation

16.28.2.1 char* build_ipv4_filter (addr_t *srcip*, addr_t *dstip*)

IPv4 filter template.

Create an IPv4 filter based on a source IP and a destination IP. The source IP is a 32bit field in the IPv4 header starting at offset 12Bytes, the destination IP is also a 32bit field starting at 16Bytes.

Parameters:

srcip Filter packets coming from this source IP (IP_ADDR_ANY for any source)

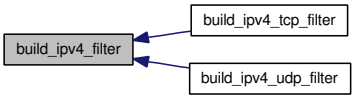
dstip Filter packets going to this destination IP (IP_ADDR_ANY for any target)

Returns:

A filter string. Caller has to free it after use.

Definition at line 154 of file tools.c.

Here is the caller graph for this function:



16.28.2.2 char* build_ipv4_tcp_filter (addr_t *srcip*, addr_t *dstip*, port_t *srcport*, port_t *dstport*)

TCP over IPv4 filter template.

This function build a TCP over IPv4 filter based on the given arguments using the build_tcp_filter and build_ipv4_filter helper functions.

Parameters:

srcip Source IP-Address to filter for (IP_ADDR_ANY for any)

dstip Destination IP-Address to filter for (IP_ADDR_ANY for any)

srcport Source TCP port to filter for (PORT_ANY for any)

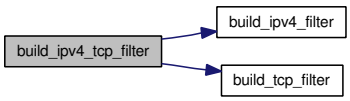
dstport Destination TCP port to filter for (PORT_ANY for any)

Returns:

A filter string. Caller has to free it after use.

Definition at line 255 of file tools.c.

Here is the call graph for this function:



16.28.2.3 char* build_ipv4_udp_filter (addr_t *srcip*, addr_t *dstip*, port_t *srcport*, port_t *dstport*)

UDP over IPv4 filter template.

This function build a UDP over IPv4 filter based on the given arguments using the build_tcp_filter and build_ipv4_filter helper functions.

Parameters:

srcip Source IP-Address to filter for (IP_ADDR_ANY for any)

dstip Destination IP-Address to filter for (IP_ADDR_ANY for any)

srcport Source UDP port to filter for (PORT_ANY for any)

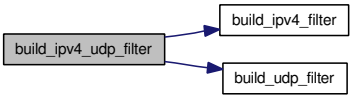
dstport Destination UDP port to filter for (PORT_ANY for any)

Returns:

A filter string. Caller has to free it after use.

Definition at line 286 of file tools.c.

Here is the call graph for this function:



16.28.2.4 char* build_tcp_filter (port_t *srcport*, port_t *dstport*)

TCP filter template.

Create a TCP filter based on the source and destination TCP Port. This filter looks for the TCP protocol number (0x06) in the IP header and sets the 16bit long source port field positioned at offset 20Bytes (with IP header) and the 16bit long destination port filed positioned at offset 22Bytes to the given arguments.

Parameters:

srcport TCP source port to filter on (PORT_ANY for any port)

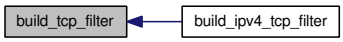
dstport TCP destination port to filter on (PORT_ANY for any port)

Returns:

A filter sting. Caller has to free it after use.

Definition at line 192 of file tools.c.

Here is the caller graph for this function:



16.28.2.5 char* build_udp_filter (port_t *srcport*, port_t *dstport*)

UDP filter template.

Create a UDP filter based on the source and destination UDP Port. This filter looks for the UDP protocol number (0x11) in the IP header and sets the 16bit long source port field positioned at offset 20Bytes (with IP header) and the 16bit long destination port filed positioned at offset 22Bytes to the given arguments.

Parameters:

srcport UDP source port to filter on (PORT_ANY for any port)

dstport UDP destination port to filter on (PORT_ANY for any port)

Returns:

A filter sting. Caller has to free it after use.

Definition at line 226 of file tools.c.

Here is the caller graph for this function:

16.28.2.6 int find_msb (uint64_t *value*) [inline]

Finds the index of the most significant 1-bit in 'value'.

Parameters:

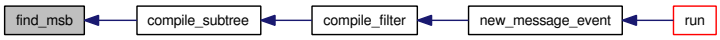
value The integer to be analyzed

Returns:

The index of the most significant 1-bit in value (bits numbered 1..64); 0 if value = 0.

Definition at line 23 of file tools.c.

Here is the caller graph for this function:



16.28.2.7 char* get_error_position_string (int *pos*)

Returns a string with pos-1 spaces and a '^' character. Used to indicate error position in filter string!

Parameters:

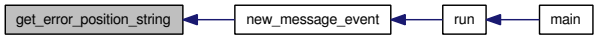
pos The position to point at

Returns:

A string with a '^' character at the given position. Caller should free memory after use!

Definition at line 88 of file tools.c.

Here is the caller graph for this function:



16.28.2.8 uint8_t* parse_hex_input (char * *str*)

Parses a string consisting of hex digits to a byte array.

Parameters:

str The string to be parsed, e.g. "fe01abc9"

Returns:

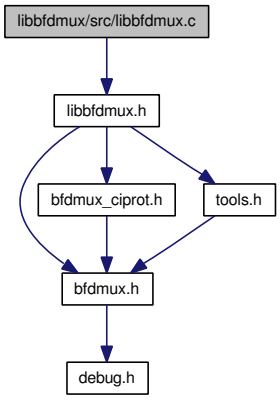
A byte array, e.g. 0xfe 0x01 0xab 0xc9. Caller should free the array after use!

Definition at line 107 of file tools.c.

16.29 libbfdmux/src/libbfdmux.c File Reference

Interface for applications that want to use bfdmux.

Include dependency graph for libbfdmux.c:



Defines

- #define CMD_BUFF_SIZE 1024

  Command interface buffer size.

- #define MQ_FLAG 0666

  Message queue permissions.

Functions

- int bfdmux_ci_send (void *data, size_t len)

  Send a command to bfdmux on the command interface.

- void * server_thread (void *ptr)

  Server thread function.

- void server_thread_start ()

  Helper function to start the server thread.

- int recv_answer (void **buffpt)

  Busy wait to wait for the answer of a sent command.

- err_t register_app (void(*recv_callback)(void *, size_t, filterid_t), void **sm_inpt, void **sm_outpt, size_t size_in, size_t size_out)

  Register application at bfdmux.

- err_t unregister_app (void)
  Deregister application from bfdmux.
- filterid_t attach (char *filter)
  Attach application to a filter.
- err_t detach (filterid_t filter_id)
  Detach application from a previous attached filter with ID 'filter_id'.
- size_t bfdmux_send (size_t len)
  Send data out.
- int bfdmux_ci_recv (void *data, size_t len)
  Receive a command from bfdmux on the command interface.
- void bfdmux_set_recv_callback (void(*callback)(void *, size_t, filterid_t))
  Set callback function for the receive packet event.

Variables

- sock_t sock
  UNIX socket (Command interface).
- mq_t mqid
  Message queue (Data interface).
- void * shmaddr_in = NULL
  Address of inbound (world to application) buffer.
- void * shmaddr_out = NULL
  Address of output (application to world) buffer.
- int smid_in
  Shared memory ID of the inbound buffer.
- int smid_out
  Shared memory ID of the outbound buffer.
- pthread_t server_thread_descriptor
  Server thread descriptor.
- void(* recv_event)(void *, size_t, filterid_t)
  Application callback function that gets called upon incoming data.

- void * request_answer

  Pointer to the answer command.

- size_t request_answer_size

  Size of the answer command.

16.29.1 Detailed Description

Interface for applications that want to use bfdmux.

This is the main file of the libbfdmux implementation. This object (and others) can be used by an application to interact with bfdmux.

Definition in file libbfdmux.c.

16.29.2 Define Documentation

16.29.2.1 #define MQ_FLAG 0666

Message queue permissions.

Todo

Verify and change if possible.

Definition at line 47 of file libbfdmux.c.

16.29.3 Function Documentation

16.29.3.1 filterid_t attach (char * *filter*)

Attach application to a filter.

Sends filter string 'filter' to bfdmux and attaches it.

Parameters:

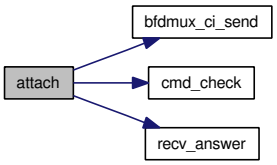
filter Pointer to the filter string

Returns:

Filter ID on success, -1 otherwise

Definition at line 326 of file libbfdmux.c.

Here is the call graph for this function:



Here is the caller graph for this function:

16.29.3.2 int bfdmux_ci_recv (void * *data*, size_t *len*)

Receive a command from bfdmux on the command interface.

Blocking wait for an incoming command on the command interface. 'len' or less command bytes will be written to 'data'.

Parameters:

→ ***data*** Pointer to a pre allocated buffer to write the command in

len Size of the the buffer

Returns:

Number of bytes written into 'data'

Definition at line 517 of file libbfdmux.c.

16.29.3.3 int bfdmux_ci_send (void * *data*, size_t *len*)

Send a command to bfdmux on the command interface.

Parameters:

data Pointer to the command

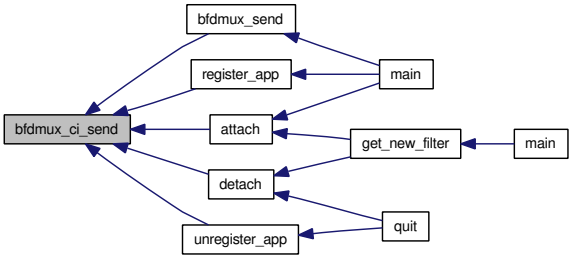
len Size of the command

Returns:

Number of bytes sent

Definition at line 491 of file libbfdmux.c.

Here is the caller graph for this function:



16.29.3.4 size_t bfdmux_send (size_t *len*)

Send data out.

Send 'len' bytes of data starting at the 'shmaddr_out' address out. The outbound buffer should be filled before calling this function.

Parameters:

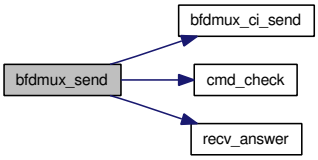
len Amount of bytes to send

Returns:

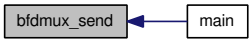
Number of bytes actually sent

Definition at line 436 of file libbfdmux.c.

Here is the call graph for this function:



Here is the caller graph for this function:



16.29.3.5 void bfdmux_set_recv_callback (void(∗)(void ∗, size_t, filterid_t) *callback*)

Set callback function for the receive packet event.

Parameters:

callback Function pointer to a receive-data handler

Definition at line 537 of file libbfdmux.c.

Here is the caller graph for this function:

16.29.3.6 err_t detach (filterid_t *filter_id*)

Detach application from a previous attached filter with ID 'filter_id'.

Parameters:

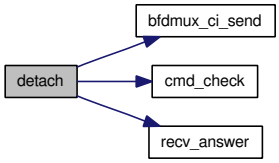
filter_id Filter ID of a previous attached filter

Returns:

OK on success, ERR otherwise.

Definition at line 384 of file libbfdmux.c.

Here is the call graph for this function:



Here is the caller graph for this function:

16.29.3.7 int recv_answer (void ** *buffpt*)

Busy wait to wait for the answer of a sent command.

Parameters:

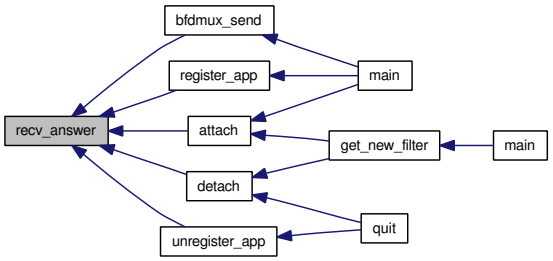
→ ***buffpt*** This pointer will point to the answer command

Returns:

Size of answer command

Definition at line 549 of file libbfdmux.c.

Here is the caller graph for this function:



16.29.3.8 err_t register_app (void(∗)(void ∗, size_t, filterid_t) *recv_callback*, void ∗∗ *sm_inpt*, void ∗∗ *sm_outpt*, size_t *size_in*, size_t *size_out*)

Register application at bfdmux.

Starts thread to handle incoming commands, sets up shared memory of specified size, connects to and registers with bfdmux via Unix socket

Parameters:

recv_callback pointer to callback function for handling incoming data

→ ***sm_inpt*** pointer to a pointer to the incoming packet buffer (will be set on success)

→ ***sm_outpt*** pointer to a pointer to the outgoing packet buffer (will be set on success)

size_in desired size of inbound buffer

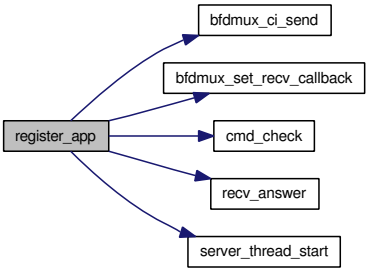
size_out desired size of outbound buffer

Returns:

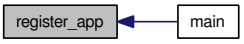
OK on success, otherwise ERR.

Definition at line 105 of file libbfdmux.c.

Here is the call graph for this function:



Here is the caller graph for this function:



16.29.3.9 void* server_thread (void * *ptr*)

Server thread function.

The server thread waits for incoming commands or answers from bfdmux and handles them separately.

Parameters:

ptr NULL

Returns:

NULL

16.29.3.10 err_t unregister_app (void)

Deregister application from bfdmux.

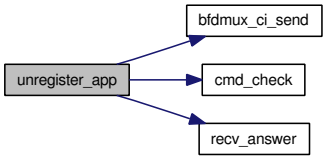
Deregisters from bfdmux, detaches shared memory and then closes command socket

Returns:

Returns OK if successful, otherwise ERR

Definition at line 252 of file libbfdmux.c.

Here is the call graph for this function:



Here is the caller graph for this function:

16.29.4 Variable Documentation

16.29.4.1 pthread_t server_thread_descriptor

Server thread descriptor.

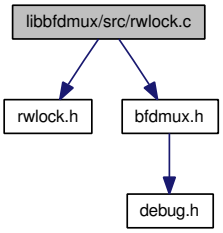
This thread listens for incoming commands from bfdmux on the command interface.

Definition at line 64 of file libbfdmux.c.

16.30 libbfdmux/src/rwlock.c File Reference

Read write lock.

Include dependency graph for rwlock.c:



Functions

- int rwlock_create (void)
  Create a read/write lock handle.
- bool rwlock_destroy (int sid)
  Destroy a previous created read/write lock.
- bool rwlock_acquire (int sid, bool write)
  Acquire rights (read-write or read-only) on an existing lock.
- bool rwlock_elevate (int sid)
  Elevate a read-only lock to a read-write lock.
- bool rwlock_try_acquire (int sid, bool write)
  Try to acquire rights (read-write or read-only) on an existing lock.
- bool rwlock_release (int sid, bool write)
  Release a previous acquired right on a read/write lock.
- bool rwlock_lower (int sid)
  Lower the permission on a read/write lock.

16.30.1 Detailed Description

Read write lock.

Read/write lock using semaphore, built after example from http://www.experts-exchange.com/Programming/Languages/C/Q_-23939132.html

Definition in file rwlock.c.

16.30.2 Function Documentation

16.30.2.1 bool rwlock_acquire (int *sid*, bool *write*)

Acquire rights (read-write or read-only) on an existing lock.

Acquire a read-write or a read-only right on a previous created read/write lock. This function call is blocking and will return only after successfully acquiring the asked right or on error.

Parameters:

sid Read/write lock id

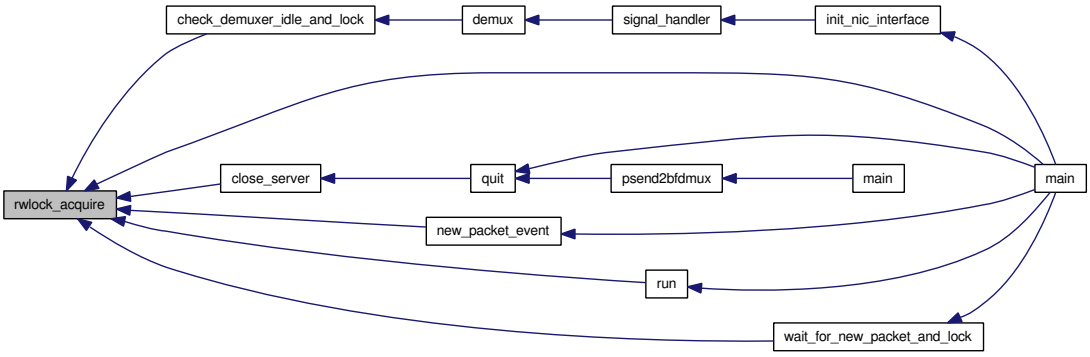
write Boolean argument to choose between a read-write lock (true) or a read-only lock (false)

Returns:

True on success, false otherwise.

Definition at line 79 of file rwlock.c.

Here is the caller graph for this function:



16.30.2.2 int rwlock_create (void)

Create a read/write lock handle.

Returns:

Read/write lock handle on success, -1 otherwise.

Definition at line 26 of file rwlock.c.

Here is the caller graph for this function:

16.30.2.3 bool rwlock_destroy (int *sid*)

Destroy a previous created read/write lock.

Parameters:

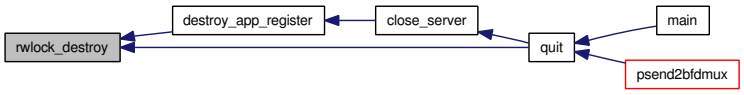
sid Previous created read/write lock that should get destroyed

Returns:

True on success, false otherwise.

Definition at line 61 of file rwlock.c.

Here is the caller graph for this function:



16.30.2.4 bool rwlock_elevate (int *sid*)

Elevate a read-only lock to a read-write lock.

Elevate the right on lock from read-only to read-write. This function is blocking and will return only after successfully elevating the read/write lock or after an error.

Parameters:

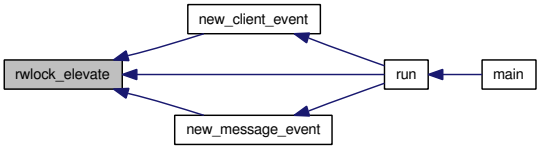
sid Read/write lock ID to elevate the rights on

Returns:

True on success, false otherwise.

Definition at line 126 of file rwlock.c.

Here is the caller graph for this function:



16.30.2.5 bool rwlock_lower (int *sid*)

Lower the permission on a read/write lock.

This function lowers the permission on a read/write lock from read-write to read-only. To completely remove the read/write lock use rwlock_release().

Parameters:

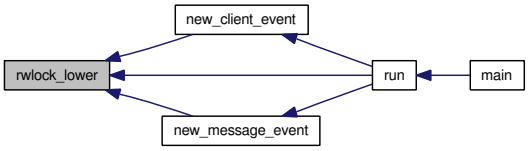
sid Read/write lock id

Returns:

True on success, false otherwise.

Definition at line 251 of file rwlock.c.

Here is the caller graph for this function:



16.30.2.6 bool rwlock_release (int *sid*, bool *write*)

Release a previous acquired right on a read/write lock.

Release a read/write lock. To lower the right from read-write to read-only don't use this function, use rwlock_lower(). This function completely releases the read/write lock.

Parameters:

sid Read/write lock id

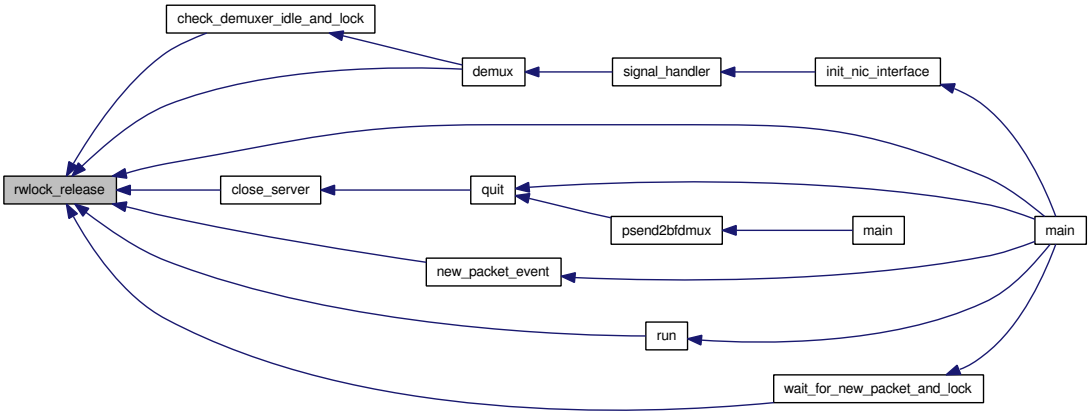
write Boolean argument specifying if this read/write lock has read-write (true) or read-only (false) permissions.

Returns:

True on success, false otherwise.

Definition at line 209 of file rwlock.c.

Here is the caller graph for this function:



16.30.2.7 bool rwlock_try_acquire (int *sid*, bool *write*)

Try to acquire rights (read-write or read-only) on an existing lock.

Acquire a read-write or a read-only right on a previous created read/write lock. This function call is non-blocking and will return immediately.

Parameters:

sid Read/write lock id

write Boolean argument to choose between a read-write lock (true) or a read-only lock (false)

Returns:

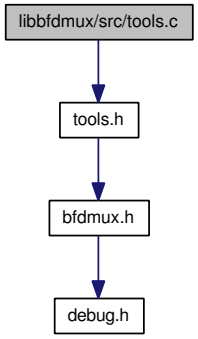
True on success, false otherwise.

Definition at line 167 of file rwlock.c.

16.31 libbfdmux/src/tools.c File Reference

Helper functoin and additional tools used by libbfdmux.

Include dependency graph for tools.c:



Functions

- int find_msb (uint64_t value)

  Finds the index of the most significant 1-bit in 'value'.

- char * get_error_position_string (int pos)

  Returns a string with pos-1 spaces and a '^' character. Used to indicate error position in filter string!

- uint8_t * parse_hex_input (char *str)

  Parses a string consisting of hex digits to a byte array.

- char * build_ipv4_filter (addr_t srcip, addr_t dstip)

  IPv4 filter template.

- char * build_tcp_filter (port_t srcport, port_t dstport)

  TCP filter template.

- char * build_udp_filter (port_t srcport, port_t dstport)

  UDP filter template.

- char * build_ipv4_tcp_filter (addr_t srcip, addr_t dstip, port_t srcport, port_t dstport)

  TCP over IPv4 filter template.

- char * build_ipv4_udp_filter (addr_t srcip, addr_t dstip, port_t srcport, port_t dstport)

  UDP over IPv4 filter template.

16.31.1 Detailed Description

Helper functoin and additional tools used by libbfdmux.

Definition in file tools.c.

16.31.2 Function Documentation

16.31.2.1 char* build_ipv4_filter (addr_t *srcip*, addr_t *dstip*)

IPv4 filter template.

Create an IPv4 filter based on a source IP and a destination IP. The source IP is a 32bit field in the IPv4 header starting at offset 12Bytes, the destination IP is also a 32bit field starting at 16Bytes.

Parameters:

srcip Filter packets coming from this source IP (IP_ADDR_ANY for any source)

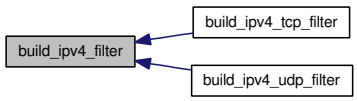
dstip Filter packets going to this destination IP (IP_ADDR_ANY for any target)

Returns:

A filter string. Caller has to free it after use.

Definition at line 154 of file tools.c.

Here is the caller graph for this function:



16.31.2.2 char* build_ipv4_tcp_filter (addr_t *srcip*, addr_t *dstip*, port_t *srcport*, port_t *dstport*)

TCP over IPv4 filter template.

This function build a TCP over IPv4 filter based on the given arguments using the build_tcp_filter and build_ipv4_filter helper functions.

Parameters:

srcip Source IP-Address to filter for (IP_ADDR_ANY for any)

dstip Destination IP-Address to filter for (IP_ADDR_ANY for any)

srcport Source TCP port to filter for (PORT_ANY for any)

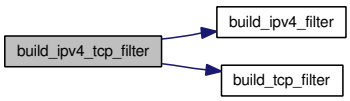
dstport Destination TCP port to filter for (PORT_ANY for any)

Returns:

A filter string. Caller has to free it after use.

Definition at line 255 of file tools.c.

Here is the call graph for this function:



16.31.2.3 char* build_ipv4_udp_filter (addr_t *srcip*, addr_t *dstip*, port_t *srcport*, port_t *dstport*)

UDP over IPv4 filter template.

This function build a UDP over IPv4 filter based on the given arguments using the build_tcp_filter and build_ipv4_filter helper functions.

Parameters:

srcip Source IP-Address to filter for (IP_ADDR_ANY for any)

dstip Destination IP-Address to filter for (IP_ADDR_ANY for any)

srcport Source UDP port to filter for (PORT_ANY for any)

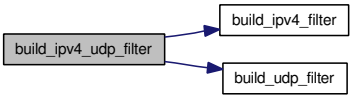
dstport Destination UDP port to filter for (PORT_ANY for any)

Returns:

A filter string. Caller has to free it after use.

Definition at line 286 of file tools.c.

Here is the call graph for this function:



16.31.2.4 char* build_tcp_filter (port_t *srcport*, port_t *dstport*)

TCP filter template.

Create a TCP filter based on the source and destination TCP Port. This filter looks for the TCP protocol number (0x06) in the IP header and sets the 16bit long source port field positioned at offset 20Bytes (with IP header) and the 16bit long destination port filed positioned at offset 22Bytes to the given arguments.

Parameters:

srcport TCP source port to filter on (PORT_ANY for any port)

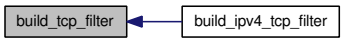
dstport TCP destination port to filter on (PORT_ANY for any port)

Returns:

A filter sting. Caller has to free it after use.

Definition at line 192 of file tools.c.

Here is the caller graph for this function:



16.31.2.5 char* build_udp_filter (port_t *srcport*, port_t *dstport*)

UDP filter template.

Create a UDP filter based on the source and destination UDP Port. This filter looks for the UDP protocol number (0x11) in the IP header and sets the 16bit long source port field positioned at offset 20Bytes (with IP header) and the 16bit long destination port filed positioned at offset 22Bytes to the given arguments.

Parameters:

srcport UDP source port to filter on (PORT_ANY for any port)

dstport UDP destination port to filter on (PORT_ANY for any port)

Returns:

A filter sting. Caller has to free it after use.

Definition at line 226 of file tools.c.

Here is the caller graph for this function:

16.31.2.6 int find_msb (uint64_t *value*) [inline]

Finds the index of the most significant 1-bit in 'value'.

Parameters:

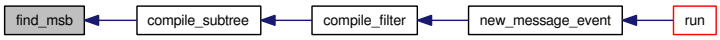
value The integer to be analyzed

Returns:

The index of the most significant 1-bit in value (bits numbered 1..64); 0 if value = 0.

Definition at line 23 of file tools.c.

Here is the caller graph for this function:



16.31.2.7 char* get_error_position_string (int *pos*)

Returns a string with pos-1 spaces and a '^' character. Used to indicate error position in filter string!

Parameters:

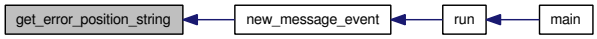
pos The position to point at

Returns:

A string with a '^' character at the given position. Caller should free memory after use!

Definition at line 88 of file tools.c.

Here is the caller graph for this function:



16.31.2.8 uint8_t* parse_hex_input (char * *str*)

Parses a string consisting of hex digits to a byte array.

Parameters:

str The string to be parsed, e.g. "fe01abc9"

Returns:

A byte array, e.g. 0xfe 0x01 0xab 0xc9. Caller should free the array after use!

Definition at line 107 of file tools.c.

Index

add_app
    register.c, 123
    register.h, 105
add_filter
    register.c, 123
    register.h, 106
attach
    libbfdmux.c, 180
    libbfdmux.h, 164

bfdmux.c
    check_demuxer_idle_and_lock, 77
    demux, 77
    main, 78
    queue, 80
    queue_first, 80
    quit, 79
    server_thread, 81
    wait_for_new_packet_and_lock, 80
bfdmux.h
    demux, 154
bfdmux/ Directory Reference, 41
bfdmux/src/ Directory Reference, 56
bfdmux/src/bfdmux.c, 75
bfdmux/src/codegen.c, 82
bfdmux/src/filter.c, 87
bfdmux/src/include/ Directory Reference, 47
bfdmux/src/include/codegen.h, 90
bfdmux/src/include/filter.h, 93
bfdmux/src/include/netif.h, 98
bfdmux/src/include/netif/ Directory Reference, 51
bfdmux/src/include/netif/mqif.h, 101
bfdmux/src/include/opdefs.h, 102
bfdmux/src/include/register.h, 104
bfdmux/src/include/server.h, 109
bfdmux/src/include/vm.h, 113
bfdmux/src/netif/ Directory Reference, 50
bfdmux/src/netif/mqif.c, 115
bfdmux/src/opdefs.c, 119
bfdmux/src/register.c, 122
bfdmux/src/server.c, 126
bfdmux/src/vm.c, 134
bfdmux_ci_recv
    libbfdmux.c, 181
bfdmux_ci_send
    libbfdmux.c, 181
bfdmux_ciprot.c
    cmd_check, 149
    cmd_get, 150
    cmd_get_size, 150
bfdmux_ciprot.h
    cmd_check, 158
    cmd_get, 159
    CMD_GET_IP_LIST, 157
    CMD_GET_IP_LIST_ANSWER, 158
    cmd_get_size, 159
    CMD_RECV, 158
bfdmux_send
    libbfdmux.c, 182
    libbfdmux.h, 164
bfdmux_set_recv_callback
    libbfdmux.c, 182
    libbfdmux.h, 165
bfdmuxchat.c
    main, 137
    new_msg, 138
    quit, 138
bfdmuxinject.c
    main, 143
    pprint, 143
    psend2bfdmux, 143
bfdmuxsniff.c
    new_msg, 146
    printicmpinfo, 146
    printipv4info, 146
    printtcpinfo, 147
    printudpinfo, 147
buf_in
    client_app, 58

buf_out
client_app, 58
build_ipv4_filter
tools.c, 193
tools.h, 174
build_ipv4_tcp_filter
tools.c, 193
tools.h, 174
build_ipv4_udp_filter
tools.c, 194
tools.h, 175
build_tcp_filter
tools.c, 194
tools.h, 175
build_udp_filter
tools.c, 195
tools.h, 176

calc
vm.c, 135
can_receive
client_app, 58
check_demuxer_idle_and_lock
bfdmux.c, 77
client_app, 57
buf_in, 58
buf_out, 58
can_receive, 58
filters, 58
close_nic_interface
mqif.c, 116
netif.h, 98
close_server
server.c, 127
server.h, 110
cmd_attach, 59
cmd_attach_answer, 60
filter_id, 60
cmd_check
bfdmux_ciprot.c, 149
bfdmux_ciprot.h, 158
cmd_detach, 61
cmd_detach_answer, 62
cmd_error, 63
cmd_get
bfdmux_ciprot.c, 150
bfdmux_ciprot.h, 159
CMD_GET_IP_LIST
bfdmux_ciprot.h, 157
CMD_GET_IP_LIST_ANSWER
bfdmux_ciprot.h, 158
cmd_get_size
bfdmux_ciprot.c, 150
bfdmux_ciprot.h, 159
CMD_RECV
bfdmux_ciprot.h, 158
cmd_recv, 64
cmd_recv_answer, 65
cmd_register, 66
cmd_register_answer, 67
cmd_send, 68
cmd_send_answer, 69
cmd_unregister, 70
cmd_unregister_answer, 71
codegen.c
compile_filter, 83
compile_subtree, 83
ensure_enough_space, 84
find_operator, 85
remove_spaces_and_braces, 85
substrfind, 86
codegen.h
compile_filter, 91
INCREMENTAL_ALLOC_SIZE, 91
INITIAL_ALLOC_SIZE, 91
MAX_FILTER_CODE_SIZE, 91
compile_filter
codegen.c, 83
codegen.h, 91
compile_subtree
codegen.c, 83

debug.h
DEBUG_LEVEL, 161
PDEBUG_RAW, 162
DEBUG_LEVEL
debug.h, 161
demux
bfdmux.c, 77
bfdmux.h, 154
detach
libbfdmux.c, 183
libbfdmux.h, 165
doc/ Directory Reference, 45

ensure_enough_space
codegen.c, 84
execute_filter
vm.c, 135

vm.h, 114

filter, 72
filter.c
get_receipient_list, 88
new_packet_event, 88
filter.h
get_receipient_list, 96
new_packet_event, 97
OP_AND, 95
OP_OR, 96
filter_id
cmd_attach_answer, 60
filters
client_app, 58
find_app_in_table
register.c, 124
register.h, 106
find_msb
tools.c, 195
tools.h, 176
find_operator
codegen.c, 85
forward_packet_to_client
server.c, 128
server.h, 110

get_error_position_string
tools.c, 196
tools.h, 177
get_receipient_list
filter.c, 88
filter.h, 96

INCREMENTAL_ALLOC_SIZE
codegen.h, 91
init_nic_interface
mqif.c, 117
netif.h, 99
INITIAL_ALLOC_SIZE
codegen.h, 91
initialize_server
server.c, 128
server.h, 111

libbfdmux.c
attach, 180
bfdmux_ci_recv, 181
bfdmux_ci_send, 181
bfdmux_send, 182
bfdmux_set_recv_callback, 182
detach, 183
MQ_FLAG, 180
recv_answer, 183
register_app, 184
server_thread, 185
server_thread_descriptor, 186
unregister_app, 185
libbfdmux.h
attach, 164
bfdmux_send, 164
bfdmux_set_recv_callback, 165
detach, 165
register_app, 166
unregister_app, 167
libbfdmux/ Directory Reference, 48
libbfdmux/bfdmuxchat/ Directory Reference, 42
libbfdmux/bfdmuxchat/bfdmuxchat.c, 137
libbfdmux/bfdmuxchat/msgq_loopback/ Directory Reference, 49
libbfdmux/bfdmuxchat/msgq_loopback/src/ Directory Reference, 55
libbfdmux/bfdmuxchat/msgq_loopback/src/msgq_clear.c, 140
libbfdmux/bfdmuxchat/msgq_loopback/src/msgq_loopback.c, 141
libbfdmux/bfdmuxsniff/ Directory Reference, 44
libbfdmux/bfdmuxsniff/bfdmuxinject/ Directory Reference, 43
libbfdmux/bfdmuxsniff/bfdmuxinject/src/ Directory Reference, 54
libbfdmux/bfdmuxsniff/bfdmuxinject/src/bfdmuxinject.c, 142
libbfdmux/bfdmuxsniff/src/ Directory Reference, 53
libbfdmux/bfdmuxsniff/src/bfdmuxsniff.c, 145
libbfdmux/src/ Directory Reference, 52
libbfdmux/src/bfdmux_ciprot.c, 149
libbfdmux/src/include/ Directory Reference, 46
libbfdmux/src/include/bfdmux.h, 152
libbfdmux/src/include/bfdmux_ciprot.h, 155

libbfdmux/src/include/debug.h, 161
libbfdmux/src/include/libbfdmux.h, 163
libbfdmux/src/include/rwlock.h, 168
libbfdmux/src/include/tools.h, 173
libbfdmux/src/libbfdmux.c, 178
libbfdmux/src/rwlock.c, 187
libbfdmux/src/tools.c, 192

main
    bfdmux.c, 78
    bfdmuxchat.c, 137
    bfdmuxinject.c, 143
MAX_FILTER_CODE_SIZE
    codegen.h, 91
MQ_FLAG
    libbfdmux.c, 180
mqif.c
    close_nic_interface, 116
    init_nic_interface, 117
    nic_send, 117
    NUM_BUFS, 116
    signal_handler, 118

netif.h
    close_nic_interface, 98
    init_nic_interface, 99
    nic_send, 99
new_client_event
    server.c, 129
new_message_event
    server.c, 129
new_msg
    bfdmuxchat.c, 138
    bfdmuxsniff.c, 146
new_packet_event
    filter.c, 88
    filter.h, 97
nic_message_buf, 73
nic_send
    mqif.c, 117
    netif.h, 99
NUM_BUFS
    mqif.c, 116

OP_AND
    filter.h, 95
op_def_t, 74
op_list
    opdefs.c, 119
    opdefs.h, 103
OP_OR
    filter.h, 96
opdefs.c
    op_list, 119
opdefs.h
    op_list, 103

parse_hex_input
    tools.c, 196
    tools.h, 177
PDEBUG_RAW
    debug.h, 162
pprint
    bfdmuxinject.c, 143
printicmpinfo
    bfdmuxsniff.c, 146
printipv4info
    bfdmuxsniff.c, 146
printtcpinfo
    bfdmuxsniff.c, 147
printudpinfo
    bfdmuxsniff.c, 147
psend2bfdmux
    bfdmuxinject.c, 143

queue
    bfdmux.c, 80
queue_first
    bfdmux.c, 80
quit
    bfdmux.c, 79
    bfdmuxchat.c, 138

read_bytes
    server.c, 130
recv_answer
    libbfdmux.c, 183
register.c
    add_app, 123
    add_filter, 123
    find_app_in_table, 124
    remove_app, 125
    remove_filter, 125
register.h
    add_app, 105
    add_filter, 106
    find_app_in_table, 106
    remove_app, 107
    remove_filter, 107
register_app

libbfdmux.c, 184
libbfdmux.h, 166
remove_app
register.c, 125
register.h, 107
remove_filter
register.c, 125
register.h, 107
remove_spaces_and_braces
codegen.c, 85
run
server.c, 131
server.h, 111
rwlock.c
rwlock_acquire, 188
rwlock_create, 188
rwlock_destroy, 189
rwlock_elevate, 189
rwlock_lower, 189
rwlock_release, 190
rwlock_try_acquire, 191
rwlock.h
rwlock_acquire, 168
rwlock_create, 169
rwlock_destroy, 169
rwlock_elevate, 170
rwlock_lower, 170
rwlock_release, 171
rwlock_try_acquire, 172
rwlock_acquire
rwlock.c, 188
rwlock.h, 168
rwlock_create
rwlock.c, 188
rwlock.h, 169
rwlock_destroy
rwlock.c, 189
rwlock.h, 169
rwlock_elevate
rwlock.c, 189
rwlock.h, 170
rwlock_lower
rwlock.c, 189
rwlock.h, 170
rwlock_release
rwlock.c, 190
rwlock.h, 171
rwlock_try_acquire
rwlock.c, 191
rwlock.h, 172
server.c
close_server, 127
forward_packet_to_client, 128
initialize_server, 128
new_client_event, 129
new_message_event, 129
read_bytes, 130
run, 131
sighandler, 132
write_bytes, 132
server.h
close_server, 110
forward_packet_to_client, 110
initialize_server, 111
run, 111
server_thread
bfdmux.c, 81
libbfdmux.c, 185
server_thread_descriptor
libbfdmux.c, 186
sighandler
server.c, 132
signal_handler
mqif.c, 118
substrfind
codegen.c, 86

tools.c
build_ipv4_filter, 193
build_ipv4_tcp_filter, 193
build_ipv4_udp_filter, 194
build_tcp_filter, 194
build_udp_filter, 195
find_msb, 195
get_error_position_string, 196
parse_hex_input, 196
tools.h
build_ipv4_filter, 174
build_ipv4_tcp_filter, 174
build_ipv4_udp_filter, 175
build_tcp_filter, 175
build_udp_filter, 176
find_msb, 176
get_error_position_string, 177
parse_hex_input, 177

unregister_app
libbfdmux.c, 185
libbfdmux.h, 167

vm.c

calc, 135
execute_filter, 135
vm.h
execute_filter, 114

wait_for_new_packet_and_lock
bfdmux.c, 80
write_bytes
server.c, 132