NAME

ng_btsocket - Bluetooth sockets layer

SYNOPSIS

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/bitstring.h>
#include <netgraph/bluetooth/include/ng_hci.h>
#include <netgraph/bluetooth/include/ng_l2cap.h>
#include <netgraph/bluetooth/include/ng_btsocket.h>

DESCRIPTION

The **ng_btsocket** module implements three Netgraph node types. Each type in its turn implements one protocol within PF_BLUETOOTH domain.

BLUETOOTH_PROTO_HCI protocol

SOCK_RAW HCI sockets

Implemented by **btsock_hci_raw** Netgraph type. Raw HCI sockets allow sending of raw HCI command datagrams only to correspondents named in send(2) calls. Raw HCI datagrams (HCI commands, events and data) are generally received with recvfrom(2), which returns the next datagram with its return address. Raw HCI sockets can also be used to control HCI nodes.

The Bluetooth raw HCI socket address is defined as follows:

```
/* Bluetooth version of struct sockaddr for raw HCI sockets */
struct sockaddr_hci {
        u_char          hci_len;      /* total length */
        u_char          hci_family;   /* address family */
                char    hci_node[32]; /* address (size == NG_NODESIZ ) */
};
```

Raw HCI sockets support a number of ioctl(2) requests such as:

SIOC_HCI_RAW_NODE_GET_STATE
: Returns current state for the HCI node.

SIOC_HCI_RAW_NODE_INIT
: Turn on "inited" bit for the HCI node.

SIOC_HCI_RAW_NODE_GET_DEBUG

Returns current debug level for the HCI node.

SIOC_HCI_RAW_NODE_SET_DEBUG
Sets current debug level for the HCI node.

SIOC_HCI_RAW_NODE_GET_BUFFER
Returns current state of data buffers for the HCI node.

SIOC_HCI_RAW_NODE_GET_BDADDR
Returns BD_ADDR for the HCI node.

SIOC_HCI_RAW_NODE_GET_FEATURES
Returns the list of features supported by hardware for the HCI node.

SIOC_HCI_RAW_NODE_GET_STAT
Returns various statistic counters for the HCI node.

SIOC_HCI_RAW_NODE_RESET_STAT
Resets all statistic counters for the HCI node to zero.

SIOC_HCI_RAW_NODE_FLUSH_NEIGHBOR_CACHE
Remove all neighbor cache entries for the HCI node.

SIOC_HCI_RAW_NODE_GET_NEIGHBOR_CACHE
Returns content of the neighbor cache for the HCI node.

SIOC_HCI_RAW_NODE_GET_CON_LIST
Returns list of active baseband connections (i.e., ACL and SCO links) for the HCI node.

SIOC_HCI_RAW_NODE_GET_LINK_POLICY_MASK
Returns current link policy settings mask for the HCI node.

SIOC_HCI_RAW_NODE_SET_LINK_POLICY_MASK
Sets current link policy settings mask for the HCI node.

SIOC_HCI_RAW_NODE_GET_PACKET_MASK
Returns current packet mask for the HCI node.

SIOC_HCI_RAW_NODE_SET_PACKET_MASK
Sets current packet mask for the HCI node.

SIOC_HCI_RAW_NODE_GET_ROLE_SWITCH
Returns current value of the role switch parameter for the HCI node.

SIOC_HCI_RAW_NODE_SET_ROLE_SWITCH
Sets new value of the role switch parameter for the HCI node.

The *net.bluetooth.hci.sockets.raw.ioctl_timeout* variable, that can be examined and set via sysctl(8), controls the control request timeout (in seconds) for raw HCI sockets.

Raw HCI sockets support filters. The application can filter certain HCI datagram types. For HCI event datagrams the application can set additional filter. The raw HCI socket filter defined as follows:

```
/*
 * Raw HCI socket filter.
 *
 * For packet mask use (1 << (HCI packet indicator - 1))
 * For event mask use (1 << (Event - 1))
 */

struct ng_btsocket_hci_raw_filter {
        bitstr_t bit_decl(packet_mask, 32);
        bitstr_t bit_decl(event_mask, (NG_HCI_EVENT_MASK_SIZE * 8));
};
```

The SO_HCI_RAW_FILTER option defined at SOL_HCI_RAW level can be used to obtain via getsockopt(2) or change via setsockopt(2) raw HCI socket’s filter.

BLUETOOTH_PROTO_L2CAP protocol

The Bluetooth L2CAP socket address is defined as follows:

```
/* Bluetooth version of struct sockaddr for L2CAP sockets */
struct sockaddr_l2cap {
        u_char    l2cap_len;    /* total length */
        u_char    l2cap_family; /* address family */
        uint16_t  l2cap_psm;    /* Protocol/Service Multiplexor */
        bdaddr_t  l2cap_bdaddr; /* address */
};
```

SOCK_RAW L2CAP sockets

Implemented by **btsock_l2c_raw** Netgraph type. Raw L2CAP sockets do not provide access to raw

L2CAP datagrams. These sockets used to control L2CAP nodes and to issue special L2CAP requests such as ECHO_REQUEST and GET_INFO request.

Raw L2CAP sockets support number of ioctl(2) requests such as:

SIOC_L2CAP_NODE_GET_FLAGS
Returns current state for the L2CAP node.

SIOC_L2CAP_NODE_GET_DEBUG
Returns current debug level for the L2CAP node.

SIOC_L2CAP_NODE_SET_DEBUG
Sets current debug level for the L2CAP node.

SIOC_L2CAP_NODE_GET_CON_LIST
Returns list of active baseband connections (i.e., ACL links) for the L2CAP node.

SIOC_L2CAP_NODE_GET_CHAN_LIST
Returns list of active channels for the L2CAP node.

SIOC_L2CAP_NODE_GET_AUTO_DISCON_TIMO
Returns current value of the auto disconnect timeout for the L2CAP node.

SIOC_L2CAP_NODE_SET_AUTO_DISCON_TIMO
Sets current value of the auto disconnect timeout for the L2CAP node.

SIOC_L2CAP_L2CA_PING
Issues L2CAP ECHO_REQUEST.

SIOC_L2CAP_L2CA_GET_INFO
Issues L2CAP GET_INFO request.

The *net.bluetooth.l2cap.sockets.raw.ioctl_timeout* variable, that can be examined and set via sysctl(8), controls the control request timeout (in seconds) for raw L2CAP sockets.

SOCK_SEQPACKET L2CAP sockets

Implemented by **btsock_l2c** Netgraph type. L2CAP sockets are either "active" or "passive". Active sockets initiate connections to passive sockets. By default, L2CAP sockets are created active; to create a passive socket, the listen(2) system call must be used after binding the socket with the bind(2) system call. Only passive sockets may use the accept(2) call to accept incoming connections. Only active sockets may use the connect(2) call to initiate connections.

L2CAP sockets support "wildcard addressing". In this case, socket must be bound to NG_HCI_BDADDR_ANY address. Note that PSM (Protocol/Service Multiplexor) field is always required. Once a connection has been established, the socket's address is fixed by the peer entity's location. The address assigned to the socket is the address associated with the Bluetooth device through which packets are being transmitted and received, and PSM (Protocol/Service Multiplexor).

L2CAP sockets support number of options defined at SOL_L2CAP level which can be set with setsockopt(2) and tested with getsockopt(2):

SO_L2CAP_IMTU
: Get (set) maximum payload size the local socket is capable of accepting.

SO_L2CAP_OMTU
: Get maximum payload size the remote socket is capable of accepting.

SO_L2CAP_IFLOW
: Get incoming flow specification for the socket. *Not implemented.*

SO_L2CAP_OFLOW
: Get (set) outgoing flow specification for the socket. *Not implemented.*

SO_L2CAP_FLUSH
: Get (set) value of the flush timeout. *Not implemented.*

BLUETOOTH_PROTO_RFCOMM protocol

The Bluetooth RFCOMM socket address is defined as follows:

```
/* Bluetooth version of struct sockaddr for RFCOMM sockets */
struct sockaddr_rfcomm {
        u_char   rfcomm_len;     /* total length */
        u_char   rfcomm_family;  /* address family */
        bdaddr_t rfcomm_bdaddr;  /* address */
        uint8_t  rfcomm_channel; /* channel */
};
```

SOCK_STREAM RFCOMM sockets

Note that RFCOMM sockets do not have associated Netgraph node type. RFCOMM sockets are implemented as additional layer on top of L2CAP sockets. RFCOMM sockets are either "active" or "passive". Active sockets initiate connections to passive sockets. By default, RFCOMM sockets are created active; to create a passive socket, the listen(2) system call must be used after binding the socket with the bind(2) system call. Only passive sockets may use the accept(2) call to accept incoming connections. Only active sockets may use the connect(2) call to initiate connections.

RFCOMM sockets support "wildcard addressing". In this case, socket must be bound to NG_HCI_BDADDR_ANY address. Note that RFCOMM channel field is always required. Once a connection has been established, the socket's address is fixed by the peer entity's location. The address assigned to the socket is the address associated with the Bluetooth device through which packets are being transmitted and received, and RFCOMM channel.

The following options, which can be tested with getsockopt(2) call, are defined at SOL_RFCOMM level for RFCOMM sockets:

SO_RFCOMM_MTU

Returns the maximum transfer unit size (in bytes) for the underlying RFCOMM channel. Note that application still can write/read bigger chunks to/from the socket.

SO_RFCOMM_FC_INFO

Return the flow control information for the underlying RFCOMM channel.

The *net.bluetooth.rfcomm.sockets.stream.timeout* variable, that can be examined and set via sysctl(8), controls the connection timeout (in seconds) for RFCOMM sockets.

HOOKS

These node types support hooks with arbitrary names (as long as they are unique) and always accept hook connection requests.

NETGRAPH CONTROL MESSAGES

These node types support the generic control messages.

SHUTDOWN

These nodes are persistent and cannot be shut down.

SEE ALSO

btsockstat(1), socket(2), netgraph(4), ng_bluetooth(4), ng_hci(4), ng_l2cap(4), ngctl(8), sysctl(8)

HISTORY

The **ng_btsocket** module was implemented in FreeBSD 5.0.

AUTHORS

Maksim Yevmenkin <*m_evmenkin@yahoo.com*>

BUGS

Most likely. Please report if found.