RPC

Enrico Deiana, Emanuele Del Sozzo

1 rpc.h

The global variables declared here are:

- extern PacketEncoder *rpcInterfaceShadowDaemonPacketEncoder
- extern PacketEncoder *rpcInterfaceTerminalPacketEncoder
- extern PacketEncoder *rpcInterfaceKernelPacketEncoder
- extern EtnNullEncoder *rpcInterfaceNullEncoder
- extern EtnBufferDecoder *rpcInterfaceBufferDecoder
- extern EtnRpcHost *rpcInterfaceShadowDaemonHost
- extern EtnRpcHost *rpcInterfaceTerminalHost
- extern EtnRpcHost *rpcInterfaceKernelHost
- extern EtnRpcHost *rpcInterfaceNullHost

Using extern keyword, C variables are declared but not defined. So, they must be defined in another header or c file before using them. In particular, these variables are defined in *packetEncoder.c* Moreover, the extern extends the visibility to the whole program. These global variables are used as hosts, encoders and decoders of Ethos primary components.

The functions exported by the header file are:

- *void rpcInit(void)*
  This function calls some other functions that initialize rpc components. The definition is inside *rpc.c*.
- *rpcCall(fn, host, connection, eventId, ...)*
  This macro is used to:
  1. calculate length of encoded packet with a dummy run
  2. reset the packet encoder with this length
  3. perform the actual RPC call

2 rpc.c

The global variable declared and defined here is:

- DebugFlagDesc debugFlagDesc[]
  This is simply an array of debug flags. It is in *rpc.c* because it is used from both *Ethos* and *Dom0* even though it is not RPC-specific.

In this file, there are some debug purpose functions. They are used to send a ping between both Shadow-daemon and Terminal. Here they are:

- *void rpcShadowDaemonPing (EtnRpcHost *h, uint64_t eventId)*
- *void rpcShadowDaemonPingReply (EtnRpcHost *h, uint64_t eventId, Status status)*
- *void rpcTerminalPing (EtnRpcHost *h, uint64_t eventId)*
- *void rpcTerminalPingReply (EtnRpcHost *h, uint64_t eventId, Status status)*

The other functions defined here are:

- *void rpcInitInterfaces(void)*
  This function initialize the interfaces. All the variables that are initialized are global variables declared in *rpc.h* and defined in *packetEncoder.c*. For each Ethos primary component a packet encoder is defined. Then, a null encoder and a buffer decoder are also defined. These five interfaces, in addiction to the servers of each primary component, are used to define the interface host of each primary component.
- *void rpcInit(void)*
  This function calls some other functions that initialize rpc components. The declaration is inside *rpc.h*.

3 packetEncoder.h

The data structure defined here is *packetEncoder*. It contains:

- EtnBufferEncoder encoder
- Packet *packet
- uint32_t totalLength
- bool packetSent
- Connection *connection

The functions declared here are:

- *PacketEncoder *packetEncoderNew(void)*
  This function creates a new, blank packetEncoder. Must be reset before use. The definition is inside *packetEncoder.c*.
- *void packetEncoderReset(PacketEncoder *e, Connection *c, uint32_t totalLength)*
  This function resets the packetEncoder. The definition is inside *packetEncoder.c*.
- *int packetEncoderWrite(EtnEncoder *_e, uint8_t *data, uint32_t length)*
  This function writes data to a packet. Inside *packetEncoder.c* there is the definition of *_packetEncoderWrite* which is *static*. By definition of *static* keyword, a function declared *static* cannot be seen outside the file it was defined in. So, if they are the same function, it is not clear why it is also defined in header file.
- *void packetEncoderFlush(EtnEncoder *e)*
  This function flushes packet encoder, sending the packet as the final fragment. Inside *packetEncoder.c* there is the definition of *_packetEncoderFlush* which is static. By definition of *static* keyword, a function declared *static* cannot be seen outside the file it was defined in. So, if they are the same function, it is not clear why it is also defined in header file.

4 packetEncoder.c

The global variables declared in *rpc.h* are defined here.
The functions defined here are:

- *static void _packetEncoderFlush (EtnEncoder *_e)*
  This function flushes packet encoder, sending the packet as the final fragment.

- *static void _packetEncoderCreateMax (PacketEncoder *e, bool firstPacket)*
  This function creates a packet of the maximum size allowed by the corresponding tunnel. It will increment sequence counter; if a packet is created here it must be sent.

- *static int _packetEncoderWrite (EtnEncoder *_e, uint8_t *data, uint32_t length)*
  This function writes data to a packet, sending fragments as maximum size reached. The final (non-full) fragment is not sent because there may be remaining data to write to it. This fragment is sent when the flush function is called. Note that if the write does not fill the first fragment, then nothing will be sent until flush is called.

- *void packetEncoderReset (PacketEncoder *e, Connection *c, uint32_t totalLength)*
  This function resets the packetEncoder. The declaration is inside *packetEncoder.h*.

- *PacketEncoder *packetEncoderNew()*
  This function creates a new, blank packetEncoder. Must be reset before use. The declaration is inside *packetEncoder.h*.

5 connection.h

This header contains the following enum structure in order to classify connection related operation codes:

```
/* Operation Codes */
enum Connection_E {
    NewConnection=0, // request for a new connection
    AcceptConnection=1, // accept an incoming connection
    RejectConnection=2, // reject an incoming connection
    CloseConnection=3, // close a connection
    SendOnConnection=4, // send on a connection a middle packet
    SendOnConnectionF=5, // send on a connection the first packet
    SendOnConnectionL=6, // send on a connection the last packet
    SendOnConnectionFL=7 // send on a connection a single packet (first and
        last packet)
};
```

Another enum structure is used for the status of a connection:

```
/* Status of a Connection */
enum ConnectionStatus_E {
    ConnectionUnused,
    ConnectionInitiated,
    ConnectionEstablished
};
```

The following constants are declared and defined:

- *static const uint32 connectionOperationShift = 29*
  Constant used for coding inside a packet an operation code;

- *static const uint32 connectionFirstPacket = 1 << 29*
  Constants used for coding inside a packet information about whether it is the first packet of a Connection or not;
- *static const uint32 connectionLastPacket = 1 << 30*
  Constants used for coding inside a packet information about whether it is the last packet of a connection or not;
- *static const uint32 connectionSizeMask = (1 << 29) - 1*

The main structure is Connection:

```
/* Represents a Connection */
struct Connection_S {
    uint32           connectionId;     // outgoing connection number, is
       tunnel specific (so, it is the same for both sides)
    ConnectionStatus connectionStatus; // status of connection
    Connection       *toConnection;    // for either split or merge, next
       connection
    uint32           next;             // linked list of incoming descriptors
    Tunnel           *tunnel;          // the tunnel the connection runs over
    Packet           *inPacket;        // an incoming packet being assembled
    RdId             ipcRdId;          // the local RdId for this connection
};
```

The header exports the following functions:

- *int mySideConnectionId(Tunnel *, uint32)*
- *void connectionProcessPacket(Connection *connection, Packet *packet)*
- *Connection *connectionAlloc(void)*
- *void connectionFree(Connection *)*
- *void acceptConnection(Connection *connection)*
- *Connection *newConnection(Tunnel *tunnel, bool firstPacket)*
- *void closeConnection(Connection *connection)*
- *void rejectConnection(Tunnel *tunnel, uint32 connectionId)*
- *void sendOnConnection(Connection *connection, Packet *packet)*
  Is declared but not defined in *connection.c* and it is no more used

6 connection.c

This file implements the exported function declared in *connection.h*:

- *int mySideConnectionId(Tunnel *, uint32)*
  Given a Tunnel and an outgoing connection number, checks whether they match (i.e. connectionId and tunnel oddSide are both even or odd)

- *void connectionProcessPacket(Connection *connection, Packet *packet)*
  Once all the fragments (network packets) have been received, they are joined into a memory packets and handled by rpcInterfaceTerminalHost, rpcInterfaceShadowDaemonHost, rpcInterfaceKernelHost respectively if the connection tunnel is associated to RpcTerminalServer, RpcShadowDaemonServer, RpcKernelServer

- *Connection *connectionAlloc(void)*
  Creates a new connection allocating the needed memory for the Connection struct

- *void connectionFree(Connection *)*
  Free the allocated memory for the given Connection

- *void acceptConnection(Connection *connection)*
  Given an incoming Connection, accepts it sending a packet with AcceptConnection code as operation and the connectionId

- *Connection *newConnection(Tunnel *tunnel, bool firstPacket)*
  Creates a new connection associated with the passed tunnel. If there is also a first packet to be sent (firstPacket = true) then it is sent with the NewConnection operation code and the new connectionId (and packet-¿firstPacket flag is set to true)

- *void closeConnection(Connection *connection)*
  Given a Connection, closes it sending a packet with CloseConnection code as operation and the connectionId

- *void rejectConnection(Tunnel *tunnel, uint32 connectionId)*
  Given an incoming Connection, rejects it sending a packet with RejectConnection code as operation and the connectionId

7 erpc.c

The main structure of **erpc.c** is EtnRpcHost, which is defined as follows:

```
/* Represents a Remote Procedure Call Host */
struct EtnRpcHost {
        EtnValue v; // value holded by the host (@see types.h)
        EtnEncoder *e; // host encoder (@see etn.h)
        EtnDecoder *d; // host decoder (@see etn.h)
};
```

The following functions are also implemented and exported by the header file *etn.h*.

- *EtnRpcHost* etnRpcHostNew(EtnValue v, EtnEncoder *e, EtnDecoder *d)*
  Given a EtnValue, EtnEncoder and EtnDecoder creates and return an EtnRpcHost

- *EtnEncoder* etnRpcHostGetEncoder (EtnRpcHost *host)*
  Given a EtnRpcHost, returns its EtnEncoder (used in kernel)

- *Status etnRpcCall(EtnRpcHost *h, EtnCallId call, EtnValue args, EtnLength *length)*
  Given EtnRpcHost (that contains the encoder to perform the rpc call), EtnCallId (that is the id of the called method), EtnValue (that are the arguments that the called method takes as input); does a rpc call and returns the Status of the call (StatusOk is returned if everything works) and the length of the encoded elements in the rpc (length of the encoded call id + length of the encoded arguments the method takes in input)

- *Status etnRpcHandle(EtnRpcHost *h)*
  Given EtnRpcHost (that contains the EtnValue v and then the method, and its arguments, to which we want perform a call), does the rpc and return the status of it

8 etn.h

The header *etn.h* exports many functions and structures, but here we show only those ones related to Remote Procedure Calls.

The main structures are EtnEncoder and EtnDecoder, which are fileds of EtnRpcHost:

```
typedef struct EtnEncoder_s {
        int (*write)(struct EtnEncoder_s *e, uint8_t *data, EtnLength length);
        void (*flush)(struct EtnEncoder_s *e);
        void *topLevelPointer; // See comments in encoder.c.
        struct rbtree addrToIndex;
        EtnLength index;
} EtnEncoder;
```

```
typedef struct EtnDecoder_s {
        int (*read) (struct EtnDecoder_s *d, uint8_t *data, EtnLength length);
        void **indexToData;
        EtnLength indexToDataLength;
        EtnLength nextIndex;
} EtnDecoder;
```

It exports also the functions defined in *erpc.c*:

- *EtnRpcHost* etnRpcHostNew(EtnValue v, EtnEncoder *e, EtnDecoder *d)*
- *EtnEncoder* etnRpcHostGetEncoder (EtnRpcHost *host)*
- *Status etnRpcCall(EtnRpcHost *h, EtnCallId call, EtnValue args, EtnLength *length)*
- *Status etnRpcHandle(EtnRpcHost *h)*

9 notes

Here the list of problems that have been found in the previous files:

- **rpc.c**
  - *void rpcShadowDaemonPingReply (EtnRpcHost *h, uint64_t eventId, Status status)*
    Parameters *h* and *eventId* are useless in this function since they are not used.
  - *void rpcTerminalPingReply (EtnRpcHost *h, uint64_t eventId, Status status)*
    Parameters *h* and *eventId* are useless in this function since they are not used.
  - *void rpcInitInterfaces(void)*
    This function contains very bad casts since they are memory allocation dependent.
- **packetEncoder.h**
  - *int packetEncoderWrite(EtnEncoder *_e, uint8_t *data, uint32_t length)*
    Inside *packetEncoder.c* there is the definition of *_packetEncoderWrite* which is *static*. By definition of *static* keyword, a function declared *static* cannot be seen outside the file it was defined in. So, if they were meant to be the same function, it is not clear why it is also defined in header file.

- *void packetEncoderFlush(EtnEncoder *e)*
  Inside *packetEncoder.c* there is the definition of *_packetEncoderFlush* which is static. By definition of *static* keyword, a function declared *static* cannot be seen outside the file it was defined in. So, if they were meant to be the same function, it is not clear why it is also defined in header file.

- **packetEncoder.c**
  - The global variables declared in *rpc.h* are defined in this file, although *rpcInterfaceNullEncoder* variable here has type *PacketEncoder*, while in file *rpc.h* it has type *EtnNullEncoder*.
  - *static void _packetEncoderFlush (EtnEncoder *_e)*
    This function contains a very bad cast since it is memory allocation dependent.
  - *static int _packetEncoderWrite (EtnEncoder *_e, uint8_t *data, uint32_t length)*
    This function contains a very bad cast since it is memory allocation dependent.
  - *void packetEncoderReset (PacketEncoder *e, Connection *c, uint32_t totalLength)*
    This function contains a very bad cast since it is memory allocation dependent.

- **connection.h**
  - The function *void sendOnConnection(Connection *connection, Packet *packet)* is declared in *connection.h* but it is not implemented in *connection.c*, nor it is used anywhere anymore.

- **connection.c**
  - In many functions defined here, *tunnelPacketCreate()* and its variants *tunnelPacketCreateEmpty()*, *tunnelPacketCreateFirst()* are often used. Since most of the code used into these functions is the same, it would be better to incorporate it into a common, generic, *packetCreate()* to improve readability and diminish lines of code.