Author: Timo Kokkonen

Document: diag

Date: 15/06/1998

Version: 0.1

# DIAG

The diag modules provides facilities to implement a uniform way of handling debug messages, information given by a software, warning, normal and fatal error messages.

## 1 Introduction

Signaling software has different needs to provide various kinds of information to the developer and the operator of the software. This modules tries to unify these needs and provide facilities that are more sophisticate than using printing to standard output or error streams directly. All other modules use this module. The design of the module is based on ideas from Linux ATM signaling software and various design patterns.

This module is inte

## 2 Architecture

Protokollan arkkitehtuurin kuvaus: spesifikaation arkkitehtuuri, luokkakaavioita jne.

Esimerkkinä kuva (*Picture*) + kuva- ja taulukkoteksti (*Caption*).

**Figure 1.** Glass of wine

### 2.1 Alikappale 1

Jokaisen otsikon jälkeen heti hieman tekstiä.

#### 2.1.1 Alialikappale 1

Alikappaleen alikappale: luultavasti tämän syvemmälle ei tarvitse mennä.

# 3 Implementation details

Informal information about the current events of the protocol framework (trace) is needed. The framework can provide that by the following debug additions.

State changes
pfProtocol::changeState can inform about state changes. The method can find out the name of the current and the new state using the RTTI. The following trace output will be displayed. This is only applicable to protocol conduits.

```
DEBUG: (ID) CHANGING_STATE: (OLD_STATE) -> (NEW_STATE)
```

Receiving trace
pfProtocol::accept can inform about received events. The method can find out the name of the received primitive using the RTTI. The following trace output will be displayed.

```
DEBUG: (ID) RECEIVING: (PROTOCOL) receives (PRIMITIVE)_(SN)
```

Sending trace
pfProtocol::toA or pfProtocol::toB can inform about sent events. The method can find out the name of the sent primitive using the RTTI. The following trace output will be displayed.

```
DEBUG: (ID) SENDING: (PRIMITIVE)_(SN)
```

Running trace
PfProtocol::runCallBack can inform about events executed events. The start and the stop of the run in identified. The method can find out the name of the executed primitive using the RTTI. The following trace output will be displayed. This is only applicable to protocol conduits.

```
DEBUG: (ID) START_RUNNING: (PROTOCOL) at (STATE) runs (PRIMITIVE)_(SN)
DEBUG: (ID) END_RUNNING
```

Protocol trace
PfProtocol can inform about the creation and deletion of a new protocol. The following trace output will be displayed.

```
DEDUG: (ID) PROTOCOL_CREATED: (PROTOCOL)
DEBUG: (ID) PROTOCOL_DELETED: (PROTOCOL)
```

User trace
User can provide additional debugging output by using a debug method in the trace object. Several choices of the format of the output are available. Variable length parameter lists are not used. The following trace output will be displayed.

```
DEBUG: (ID) TRACE: (USER PROVIDED OUTPUT) : (FILE) : (LINE)
DEBUG: (ID) TRACE: (OUTPUT) = pfULong : (PFULONG VALUE) : (FILE) :
(LINE)
DEBUG: (ID) TRACE: (OUTPUT) = string : (STRING VALUE) : (FILE) : (LINE)
DEBUG: (ID) TRACE: (OUTPUT) = pfFrame [(LENGTH)] : (FILE) : (LINE)
DEBUG: (ID) TRACE: 65 65 65 65 65 65 65 65 65 65 : AAAAAAAAAA
DEBUG: (ID) TRACE: END_OF_FRAME
```

ID = LxCxxxxxxxxxPxxxxxxxxxxMxx, where L is a link indentifier, C is a call identifier, P is a primitive identifier and M is a module identifier (two aphanumeric characters).

Errors and warnings
Protocol framework can inform about thrown exceptions and exceptions that are cached by the framework in the pfProtocol::runCallback. The following error output will be displayed.

```
ERROR: EXCEPTION_THROWN: (EXCEPTION)
ERROR: EXCEPTION_CATCHED: (EXCEPTION)
```

Module identifiers are the following.

| | | |
|---|---|---|
| CC | cc | cc |
| UNI 3.1 | uni | u3 |
| UNI 4.0 | uni40 | u4 |
| DSS2 | dss2 | d2 |
| BISUP | bisup | bi |
| SSCOP | sscop | ss |
| UNI-SSCF | usscf | us |
| NNI-SSCF | nsscf | ns |
| CPCS | cpcs | cp |
| TCAP | tcap | tc |
| MTP3 | mtp3 | m3 |
| SCCP | sccp | sc |
| GSMP | gsmp | gs |
| ILMI | ilmi | il |
| MGMT | mgmt | mg |
| SWITCH | switch | sw |
| TESTING | testing | te |
| SF | sf | sf |
| PF | pf | pf |

If more formal user trace is needed that can be provided in the following form.

Method call

```
DEBUG: (ID) TRACE: METHOD_START: CLASS = (CLASS) METHOD = (METHOD)
DEBUG: (ID) TRACE: METHOD_END: CLASS = (CLASS) METHOD = (METHOD)
```

Object creation (how about copy constructor and clone)

```
DEBUG: (ID) TRACE: OBJECT_CREATED: CLASS = (CLASS) ID = (SN)
DEBUG: (ID) TRACE: OBJECT_DELETED: CLASS = (CLASS) ID = (SN)
```

The error detection could be based on the values of UNI/DSS2/BISUP Cause information element. An object could be implemented that could raise an exception carrying information about the cause value of the error. Only one or few different exceptions are needed.

The supported cause values are the following.

Normal events

| | |
|---|---|
| 1 | unallocated (unassigned) number |
| 2 | no route to specified transit network |
| 3 | no route to destination |
| 16 | normal call clearing |
| 17 | user busy |
| 18 | no user responding |
| 21 | call rejected |
| 22 | number changed |
| 23 | user rejects all calls with calling line identification restriction |
| 27 | destination out of order |
| 28 | invalid number format (address incomplete) |
| 30 | response to STATUS ENQUIRY |
| 31 | normal, unspecified |

Resource unavailable

| | |
|---|---|
| 35 | requested VPCI/VCI not available |
| 36 | VPCI/VCI assignment failure |
| 37 | user cell rate not available |
| 38 | network out of order |
| 41 | temporary failure |
| 43 | access information discarded |
| 45 | no VPCI/VCI available |
| 47 | resource unavailability, unspecified |

Service option not available

49 Quality of Service unavailable
57 bearer capability not authorized
58 bearer capability not presently available
63 Service option not available, unspecified

Service option not implemented

65 bearer capability not implemented
73 unsupported combination of traffic parameters
78 ALL parameters cannot be supported

Invalid message

81 invalid call reference value
82 identified channel does not exist
88 incompatible destination
89 invalid endpoint reference
91 invalid transit network selection
92 too many pending add party requests

Protocol error

96 mandatory information element is missing
97 message type non-existent or not implemented
99 information element non-existent or not implemented
100 invalid information element contents
101 message not compatible with call state
102 recovery on timer expiry
104 incorrect message length
111 protocol error, unspecified

The error messages should identify the source of the error and in case of call control and network interface signaling protocol the cause value of the error. At the same time when the error diagnostics are logged, and exception is thrown. That exception is cached in a place where a call can be released. In an optional places the exception can be cached without releasing the call.

Diagnostics itself cannot throw exceptions which are propagated to call control or signaling protocols.

Perceived severity defines six severity levels, which provide an indication of how critical the event notification is.

- Critical severity level indicates that a service affecting condition has occurred and an immediate corrective action is required. This corresponds to the fatal level in Linux ATM signaling.

- Major severity level indicates that a service affecting condition has occurred and an urgent corrective action is required. This corresponds to the error level in Linux ATM signaling.

- Minor severity level indicates that the existence of a non-service affecting fault condition and that corrective action should be taken in order to prevent a more serious fault. This corresponds to the error level in Linux ATM signaling.

- Warding severity level indicates the detection of a potential or impending service-affecting fault, before any significant effects have been felt. Action should be taken to further diagnose and correct the problem in order to prevent it from becoming a more serious service-affecting fault. This corresponds to warning level in Linux ATM signaling.

Diagnostic facilities classify the diagnostic messages to five different categories. These categories are DEBUG, INFORMATION, WARNING, ERROR, and FATAL.

The use of these categories is the following.

- Debug is used for information that is useful in the development time, but it is not needed after the software is completed and can be turned off. Debug information is usually used to show that certain program code is reached and to show values of certain variables at certain point. Examples: “object b created”, “method c called”, “variable d has value d1”, “parameter e has value e1”, “primitive f arrived”, “pdu g sent” and “timer h expired”.

- Information is used for information that is useful to the operator of the software in the normal operation of the software. This is usually used when the software is started or stopped or when configuration information is loaded. Examples: “signaling started”, “port b added” and “address c removed”.

- Warning is used for information that is given when something unexpected has happened but the current operation can be completed using default values or otherwise. Some optional service maybe not be provided. A connection or call is not cleared because of the situation that caused the warning to be issued. Examples: “unrecognized information element” and “primitive b received in state c”.

- Error is used for information that is given when something unexpected has happened and the current operation cannot be completed. Even mandatory service cannot be provided. A connection or a call is cleared after an error recovery does not succeed. Examples: “error in a mandatory part of a pdu”, “using default case label”, and “communication error”

- Fatal is used for information that is given when a severe error has occurred and the continuing the operation is impossible. This class should be used in extremely rare cases and usually just in the starting phase of the software. Example: “error in configuration file”.

Different error codes should be standardized:

Object cannot be created
Object cannot be deleted
Function call failed
System call failed
Invalid parameter
Unexpected type
Unexpected value
Out of range
Assigning to itself

Not implemented
Null pointer

Alignment not succesful
Protocol error
Error in sequence numbers

Out of service
Retry counter reached
Timer expired

Encode error
Decode error
PDU in wrong protocol
Unexpected PDU in this state
Invalid message type
PDU length violation
PDU too short
PDU too long

Lack of credit

Open error
Writing error
Reading error

Error in address
Routing failure

Dynamic entity not found
Static entity not found

Congestion detected
Memory full
Memory cannot be released
Table full

# 4 Features implemented

Mitä ominaisuuksia on toteutettu ja mitä jätetty toteuttamatta?

- Ominaisuus 1. Seuraa tarpeellinen kuvaus.
- Ominaisuus 2. Ja sen kuvaus.

# 5 Known bugs and flaws

Onko versiossa bugeja tai puutteita? Milloin järjestelmä ei toimi kunnolla? Voitaisiinko joku tehdä paremmin, jos olisi aikaa?

- Bugi 1. Bugin kuvaus.
- Bugi 2. Kuten edellä.

# 6 Future development

Modulin jatkokehityssuunnitelmat. Voi sisältää aikatauluja tms., mutta lähinnä (mielestäni) ideoita siitä, miten modulia kannattaisi parantaa/laajentaa tai mitä ominaisuuksia kannattaisi lisätä.

# 7 References

Tänne viitteitä lähteisiin papereiden tyyliin.

# 8 Appendixes

Appendix A Linux ATM diagnostics examples

In the following there are examples of the use of diagnostic facilities in the Linux ATM signalling software.

Debug

| | |
|---|---|
| io.c: | “TO KERNEL: %s (%d) for 0x%lx/0x%lx <%d>” |
| proto.c: | “freeing socket 0x%lx@%p” |
| proto.c: | “socket 0x%lx enters state %s (Q.2931 %s)”, |
| q2931.c: | “AAL type %ld" |
| q2931.c: | “ITF.VPI.VCI: %d.%d.%d” |
| q2931.c: | “Incoming call from %s” |
| q2931.c: | “Cause %d (%s)” |
| timeout.c: | “T309 has expired” |

Information

| | |
|---|---|
| atmsigd.c: | “Linux ATM signaling” |
| atmsigd.c: | “Acting as %s side”,net?”NETWORK”:”USER” |
| kernel.c: | “Active close (CR 0x%06X)” |
| kernel.c: | “Added local ATM address %s at” |
| q2931.c: | “Active open succeeded (CR 0x%06X” |
| q2931.c: | “Passive close (CR 0x%06X)” |

Warning

| | |
|---|---|
| atmsigd.c: | “Not found. Using defaults.” |
| io.c: | “bad signaling write: wanted %d, wrote %d” |
| io.c: | “local address table overflow” |
| kernel.c: | “invalid message %d” |
| kernel.c: | “message %s is incompatible with state %s” |
| q2931.c: | “STATUS %s received in state %s” |
| q2931.c: | “Bad Q.2931 message %d” |
| timeout.c: | “Trouble: T308_2 has expired” |

Error

| | |
|---|---|
| atmsigd.c: | “chdir %s: %s” |
| io.c: | “read kernel: %s” |
| io.c: | “bad kernel write: wanted %d, wrote %d” |
| kernel.c: | “no local address” |
| proto.c: | “socket 0x%lx has non-empty listen queue” |
| proto.c: | “ignoring duplicate VPCI %d (itf %d)” |

q2931.c: “can't parse message - aborting the call”
q2931.c: “q_close returned <0 in to_q2931”
sap.c: “unsupported traffic class %d\n”
sap.c: “AAL type %d requested”

Fatal

atmsigd.c: “Error in config file. - Aborting.”
atmsigd.c: “fork: %s”
io.c: “kernel message too short (%d < %d)”
io.c: “kernel message too big (>= %d)”