MSNL Connection Management
Using the Ultrix MSNL master

Simon Crosby

15th March 1994

1 Introduction

This document describes the Ultrix implementation of the msnl management architecture proposed in [1]. Elsewhere in this document collection the design of the MSNL control architecture and the implementation of the Unix based MSNL master is described. This document describes how the Unix MSNL master is structured internally, and how it should be used. This is not an architectural document – it serves to document the expected behavior of the MSNL master and the environment required to run it.

The MSNL master is a user space daemon which runs on an Ultrix 4.3 kernel or on OSF/1 (still under development), which has compiled into it the latest release of the MSNL user plane and socket code written by Richard Black. The MSNL master is responsible for all msnl management and control plane functions for the host on which it is supported. Internally the MSNL master is structured in such a way as to permit it to perform msnl connection management for more than 1 entity. Here an entity is taken to mean a Unix host or a wanda kernel which wishes to use msnl. The internal structure is imperfect, but largely follows the design principles described in [1]. The current version is based on an initial, experimental implementation which uses msrpc to describe a protocol used internally within the manager. This protocol is, in fact, the signalling protocol described in [1]. The use of msrpc to describe the protocol has made porting the manager somewhat awkward, because the msrpc tools are not available on all platforms and because msrpc has been superseded by msrpc2. The Unix MSNL manager should thus be rewritten in a cleaner way.

It is intended to port/rewrite the manager for wanda. When this is done, it will be possible to use the signalling protocol, currently used internally in the manager, as a generic signalling protocol for MSNL.

2 Structure of the MSNL Master

The MSNL master is responsible for implementing all functions required to set up and maintain a data path (a liaison) between two MSNL Service Access Points (SAPs) in an MSNL internetwork, and includes the following functions:

1. Signalling to request connection setup,

2. Topology determination and routeing of internetwork requests.

3. Allocation and management of MSNL SAPs.

4. Resource management and monitoring to ensure that QOS constraints are met.

5. Network maintenance functions.

The MSNL master is structured internally as two independent state machines representing the user side of the UNI and the network side of the UNI respectively. The UNI describes the protocol used by MSNL entities to request connection setup, teardown and other facilities. The full protocol has been documented and will be made available on request (it is too long to include here).

The user side of the UNI is responsible for the maintenance of all state associated with the MSNL SAP for the requesting user. This is maintained in a structure called the `fsm`, representing a finite state machine for the user side of the connection. No MSNL management or control plane functions are located on the user side of the UNI. The user side of the UNI encodes UNI requests in response to socket layer requests received from the Unix platform supporting the requesting user. The Protocol used for communication between the kernel and master is documented elsewhere in this document collection. Typical requests include requests to allocate a SAP, make a connection, offer a SAP for incoming connections, accept an incoming connection, and tear down a connection.

Requests encoded by the user side of the UNI are "transmitted" to the network side of the UNI by placing them in an asynchronous queue which is serviced by an independent thread which operates on the network side of the UNI. On the network side of the UNI each connection is referred to by its unique call reference, assigned when the SAP for the connection was first requested. Significantly more state must be maintained for connections at the network side of the UNI. In addition the full control plane functions for the MSNL protocol stack are implemented on the network side of the UNI. Messages from the Unix kernel pertaining to the state of associations, receipt of data for associations which do not exist, and other status information are managed by the association layer management, located within the control plane of the network side of the UNI. Metasignalling requests, routed from the kernel to the MSNL master, are routed to a protocol translation package which converts metasignalling into NNI requests, thereby simulating the receipt of connection requests from a remote MSNL master. Responses from the network side of the UNI to the user side are returned via a second asynchronous queue. On the network side of the UNI independent threads manage communication with the Unix kernel, the user side of the UNI, and manage timeouts for the MSNL signalling protocol. On the user side of the UNI two threads operate, one receives requests the socket layer of the Unix kernel, and the other awaits responses from the network side of the UNI.

The upper layers of the MSNL master are designed be platform independent. A platform specific layer interfaces to the host operating system. For Unix this interface is described elsewhere in this collection. The Unix interface consists of a special control socket which is used to transfer messages between the MSNL master and the kernel. On the MSNL master side of this interface, a hash table is used to map the Unix kernel notion of a call reference, namely a pointer to a protocol control block, to an integer which represents the call reference for the MSNL master. Messages passed up from the kernel are translated from the Platform Specific Interface (PSI) message format into the format expected by the MSNL master on the user side of the UNI. The UNI, PSI and NNI are described in an interface description language, MSDR, which is used by MSRPC to define service interfaces. This will be changed in the second release of the manager, planned for the summer of 1994. At this time the manager will be modified to either use msrpc2 or to use a hand-coded representation of the messages which are transferred across the interfaces described in the architectural document.

3 The Rarp Server

The MSNL master also contains a rarp server which responds to rarp requests from any of its attached MSNL interfaces. The responses which the master gives to rarp requests are described in a configuration file which is supplied to the master when it is started. Rarp requests are serviced if they are received from an Ethernet interface or if the local system is running a boot server and the request yields a response which indicates that the requester should boot from the boot server on the local system. The master will not respond to rarp requests from interfaces other than the Ethernet unless the local system will boot the requester.

4 Running the MSNL Master

The MSNL master is an Ultrix program which requires the pthreads library in order to compile and run. Internally the master is multi-threaded. The master must be run with root privileges, to enable it to open and use the MSNL control socket. It also needs to be configured with several environment variables which describe where the master can obtain information on the configuration of the system which it is managing, where it should write its log, and what responses it should deliver when responding to rarp requests. The environment variables which are required are:

- `MSNL_MASTER_LOG`: the full path them of the log file to which the tracing information printed by the master will be logged. This file will never exceed 10000 lines in length. It is periodically closed and restarted.
- `MSNL_MASTER_CONFIG`: the name of the configuration file which the master will read on start up and whenever it is signaled to reread its configuration and rarp tables (see below).
- `MSNL_MASTER_RARPD`: the full path name of the file containing the rarp table for the rarp daemon. If this is a non existent file, or does not contain valid information then the rarp daemon will not start. This will not affect the behavior of the master in any other way. This is the correct way to prevent the rarp daemon from starting. This file will be reread when the signal (described below) is given to the master to get it to reconfigure.
- `MSNL_MASTER_BOOT`: the MSNL address of the boot server which should be configured as the default boot server for all rarp requests which do not have a boot server address encoded in the rarp table. The format of this variable is an MSNL address in Internet dotted notation.

To reconfigure the master, the appropriate files should be edited (for example the rarp table) and the manager reinitialised by sending it the signal `SIGHUP`. On receipt of this signal the rarp tables will be discarded and the MSNL master will reread both the configuration file and the rarp tables. An error in either file will have the same results as if the master had just been started, namely an error in the configuration file will cause the master to fail, close the control socket and exit. An error in the rarp table will prevent the rarp daemon from starting.

5 File Formats

The two files read by the MSNL master are the configuration file and the rarp table. Their formats are described below. In both files, any line beginning with # is regarded as a comment. In the rarp table any line with a # in it is considered valid only for the part of the line proceeding the #.

5.1 MSNL_MASTER_CONFIG

This file describes the entities which the master is managing. Currently only one entity may be configured for a single master. The following example documents the file format:

```
# msnl.conf for sirius generated on Thu Mar 10 10:22:48 GMT 1994
#
# The entity being configured.
# Format of line is
# primary <msnladdress> <MSNLname> <number of pods>
#
primary 128.232.65.42   sirius-p.cl.cam.ac.uk   1
#
# Devices (pods) attached to the entity
# Format of lines is
# pod <number> <name> <msnladdress> <class> <type> <hwaddress>
#
pod 0   ln0     128.232.65.42   1 2     read-by-ioctl
pod 1   yes0    128.232.129.42  5 7     none#
#
# Routing information for each of the pods
# Format of lines is
# route <name> <net>-<mask>[,....]
#
route ln0       0.0.0.0-0.0.0.0
route yes0      128.232.128.0-255.255.240.0
```

5.2 MSNL_MASTER_RARPD

This file contains rarp response information, for two kinds of rarp requests, namely ETHER rarps and FSN (SERIAL) rarps. Each entry takes 1 line of the file. The fields are documented in the example which follows.

```
#
# $Id: rarptable,v 1.1 1993/11/22 14:51:06 rjb17 Exp rjb17 $
#
# 68k systems
#
# type   ethernet address        msnl address   booter address  # more comment
ETHER   08:00:3e:20:49:03       128.232.68.20                   # lamprey
ETHER   08:00:3e:20:45:a2       128.232.68.21                   # lumpfish
```

```
#
# FPC3 switches
#
# type  FSN id                       msnl address   booter address  # more comment
SERIAL  34                          128.232.133.131 128.232.65.27   # king03
SERIAL  55                          128.232.133.83  128.232.65.27   # duke03
```

References

[1] Simon Crosby, "MSNL Connection Management", ATM Document Collection, University of Cambridge Systems Research Group Technical Note, February 1993.