

Reboot Log

Local application BOOT

Robert Goodwin
Thu, Sep 11, 2008

To build a record of front ends that boot, local application `BOOT` captures the “comment” alarm message that each node multicasts as it comes up after reset. This note describes how this is done.

Each front end system listens to all Classic protocol messages, including alarm messages. Classic alarm messages are normally multicast to a target node# in the range `09F0–09FE`. The one to use is installed in the 4th word of the `PAGEM` system table in each node. One could write code that listens to the Classic protocol port in an arbitrary client, but a front end system code already listens to this port; hence, the same datagrams are not accessible to a local application.

What does the system code do with alarm messages that it receives? Support is included for optionally emitting alarm messages via the serial port and/or the bottom line of the “little console,” according to the setting of universal Bits `00A4`, `00A3`, and `00A2`. The implementation is built on an alarm message block that the `Alarms` task normally allocates when it prepares an alarm message for delivery to the network. After delivery, the block is freed by the `FREEBLK` code in the `QMonitor` task. For alarm messages (sent via multicast) that are received by a node, the message is placed in a message block which is placed into the same queue that is used for delivery of alarm messages to the network. But it is marked so that the code that builds datagrams will ignore it. (This queue is referred to in the source code as `OUTPQ`, or `ETHPQ`.) Whether sent to the network or not, all message block entries in this queue are reviewed by `QMonitor`.

At the point in `FREEBLK` where the message block is no longer needed, the above Bits are checked to see whether an ascii message is to be constructed for serial port delivery. In addition, for local alarms only, the relevant information is extracted from the message block and passed via an internal message queue to the local application `AERS`, whose job is to shepherd alarm delivery to the Acnet alarm handler `AEOLUS`. The new support needed for `BOOT` is modeled after this `AERS` support. For any system reset comment alarm message, whether for the local node or not, the same information is extracted and passed to another message queue monitored by `BOOT`.

An entry found in the message queue by `BOOT` is merely formatted into an 8-byte record to write to the `BOOTLOG` data stream, for which the node and data stream index are given by parameters to `BOOT`. Upon initialization, `BOOT` ensures that the target data stream is named `BOOTLOG`. Note that the `BOOTLOG` data stream does not have to reside locally.

Each front end can listen to only 8 multicast addresses, so one front end running `BOOT` is not enough to see all alarm messages that are divided across as many as 15 multicast addresses. Three instances of `BOOT` will be needed, each one in a separate node that is configured to listen to several multicast addresses. But each instance can target the same `BOOTLOG` data stream, so that listing the contents of `BOOTLOG` can be all-inclusive.

The initial implementation of `BOOT` is installed in `node0509`, which listens only to the multicast address for node `09F1`, which is used by all Linac front ends. When a Linac front end comes up after reboot, `node0509` sees it, passes it to `BOOT`, which then targets `BOOTLOG` in `node0562`.

After testing, `BOOT` was installed in nodes `0508`, `0509`, and `0562`. Each node listens to alarm messages sent to 5 different multicast addresses, so that the maximum of 15 are monitored. Each node writes a record about a node’s booting in the `BOOTLOG` data stream in `node0562`.