Fermilab Tevatron Alarms Processing System

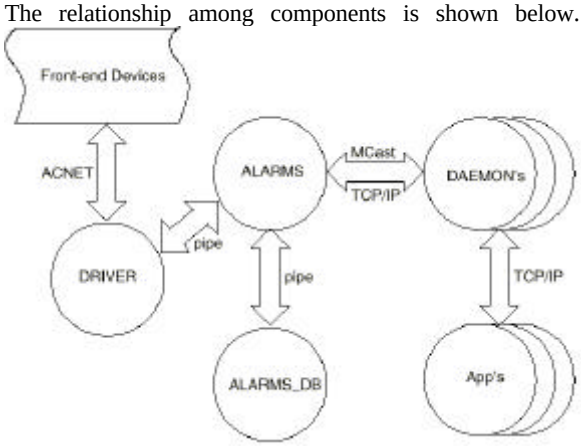
Seung-chan Ahn
Fermi National Accelerator Laboratory
PO Box 500
Batavia, IL 60510, USA

Abstract

A new distributed alarms processing software for the Fermilab Tevatron is described. The software consists of 4 components, each with clearly defined roles. They are ALARMS_DRIVER, ALARMS, ALARMS_DB and ALARMS_DAEMON. ALARMS_DRIVER handles all communications with the front-ends via Fermilab developed ACNET protocol. Additional ALARMS applications are being constructed using Java.

1 Overview of the system

A new distributed ALARMS system for the Fermilab Tevatron processes alarm messages from various devices that collectively control the Tevatron operation. The ALARMS system can be approximately divided into 4 components. Each of them performs one of the following tasks: (1) interaction with devices, (2) database look-up for device properties, (3) build alarm messages for consumers, and (4) dispatch messages to interested parties. Additionally, a Java program was written to display alarms. The relationship among components is shown below.



The distributive nature of ALARMS system is replicated in each component with an extensive use of threads. The exception is ALARMS_DB, which communicates only with ALARMS. The utilization of threads helps write cleaner and more readable code, but it also demands careful design of the software to maintain data consistency among different processes and threads within a process. In the following each component of the ALARMS system is described.

2 Alarms_driver

ALARMS_DRIVER is the component that directly communicates with front-end nodes via Fermilab developed ACNET protocol. Only one copy of DRIVER runs on a designated node. DRIVER initiates the whole ALARMS system activities. DRIVER forks 2 subprocesses and establishes proper pipe connections between itself and subprocesses: ALARMS to process alarms, and ALARMS_DB to access the device database. DRIVER then watches for subprocesses. In rare instances of ALARMS or ALARM_DB exits, DRIVER restarts them.

At start-up time DRIVER waits for ALARMS to initiate clearing all front-end nodes. When ALARMS is prepared to process alarms data, it instructs DRIVER to request front-ends to refresh local alarm data. DRIVER waits for all front-end nodes to respond but does not wait until all front-ends to respond. Responding front-ends are immediately registered as valid nodes and subsequent alarm data from them are passed to ALARMS. DRIVER makes a few more attempts to wake up the front-ends before it gives up. However, front-ends could report to DRIVER at a later time.

3 Alarms

ALARMS process builds alarm messages based on the device alarm data and the corresponding database information. ALARMS then multicasts the message using UDP protocol for DAEMON and for itself. To facilitate the proper initialization of DAEMON processes ALARMS exchanges messages with DAEMON using TCP/IP protocol.

ALARMS process assigns a sequential number for each alarm for data consistency with other components of ALARMS system. ALARMS logs all alarm data both in its raw and processed formats. The redundancy of alarm data is justified considering the low cost of storage devices.

Several threads of ALARMS run concurrently. The alarm receiver thread is set at the highest priority and other threads including the logging thread at a normal priority. ALARMS process logs alarms data at a fixed time interval or fixed sequential number interval, whichever comes first.

ALARMS process also maintains a global shared memory backed up by a physical file. The shared memory contains the most recent 20000 alarms that are readily available for other processes.

Utility programs were written to take advantage of the shared memory and other log files. An example is to produce the alarm history data for devices or a node for a given time interval.

4 Alarms_db

ALARMS_DB process makes database queries by calling Fermilab controls library CLIB routines. CLIB caches

database information internally, hence subsequent database inquiry on the same device is very fast.

ALARMS_DRIVER notifies ALARMS_DB of any change of the device database, at which point ALARMS_DB clears the cache for affected devices. ALARMS_DB maintains valid data at all times.

5 Alarms_daemon

ALARMS_DAEMON interacts with ALARMS and application programs. Much of the internal bookkeeping of DAEMON is identical to that of ALARMS. Only components exposed to external processes are different. DAEMON runs on multiple nodes as a background process.

The startup of DAEMON is not synchronized with that of ALARMS. DAEMON establishes the data consistency with ALARMS in the following sequence. At startup DAEMON requests ALARMS for active alarms. While ALARMS is sending active alarms stored in a queue, ALARMS may have received more alarms from the front-ends. Upon receiving the first multicast message DAEMON realizes that some alarms are missing or possibly invalid because missing messages might have cleared them. DAEMON makes another request for more messages to fill up the message gap.

UDP being a connectionless protocol, some messages can possibly be lost. DAEMON detects the occurrence of missing message by checking the sequential number of each alarm message. As DAEMON discovers missing alarms it multicasts a request for catch-up of alarms. To reduce the network traffic, up to 2 DAEMONs send UDP multicast messages for an identical request. While they may be in different status of message reception, they attempt some cooperation.

DAEMON runs a thread that waits for TCP/IP connections. Upon receiving the connection DAEMON starts a new thread for the application. Communication with applications is entirely asynchronous.

Applications talk with DAEMON to send commands to front-ends. DAEMON checks validity of the requester and command, and then multicast the message. ALARMS process picks it up and passes it to DRIVER, which assembles a correct ACNET message and notifies the front-ends.

6 Status

Presently all the ALARMS components other than applications run as detached background processes under VMS operating system. At the moment only ALARMS_DAEMON is ported to Unix platforms Solaris and FreeBSD. An effort to port the entire ALARMS system to Unix is under way.

An alarm display Java program was developed using Visual J++ under NT. Another Java program is being designed for graphical presentation of alarm history data for a device or for a node. Java programs run as both applet and application.