Monitoring an IOC’s status with the “alive” record

July 17, 2014

Dohn Arms
BCDA

Rationale

Issue: Want convenient central resource that lets us see if IOC is operational, irrespective of subnet boundaries.

Solution: Use centralized heartbeat as failure detection model, with a record sending UDP heartbeats to database server.

Issue: A database of IOCs that are constantly appearing and changing will be out of date when manually managed. Want automatic system of knowing information about IOCs.

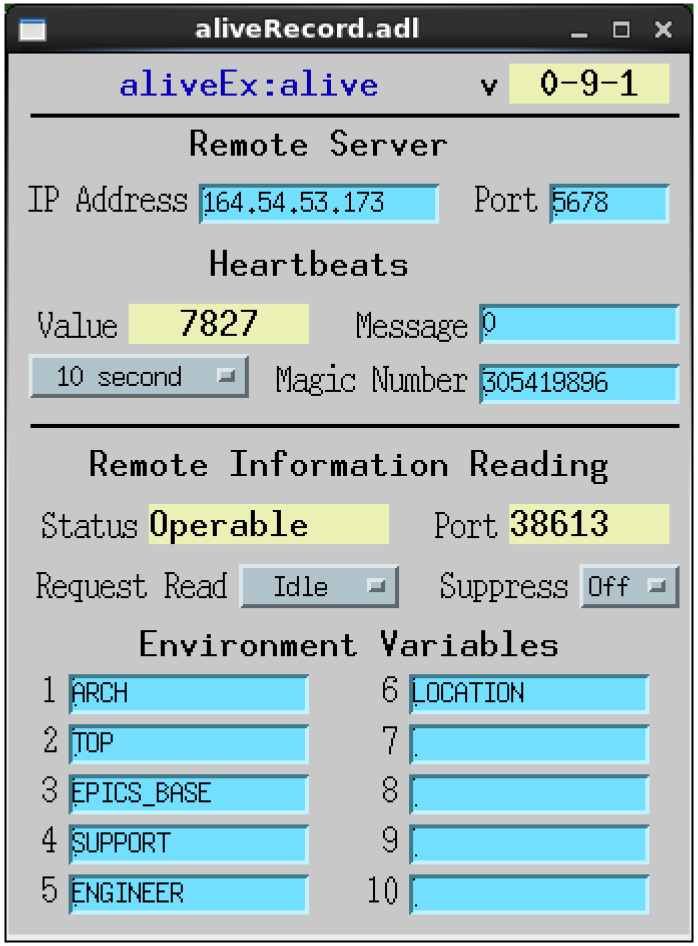
Solution: Allow the database server to query an IOC about its parameters. The IOC has a TCP port open over which it will send record-specified environment variable, as well as information relevant to the IOC type.

alive record

Uses a custom network protocol to talk to the database server.

Has two parts:

- Part that processes according to normal record rules, sending UDP heartbeats to the database server.
- Spawned thread that has an open TCP port, waiting for information requests (only from database server).



aliveRecord.adl

aliveEx:alive v 0-9-1

Remote Server

IP Address 164.54.53.173 Port 5678

Heartbeats

Value 7827 Message 0

10 second Magic Number 305419896

Remote Information Reading

Status Operable Port 38613

Request Read Idle Suppress Off

Environment Variables

1	ARCH	6	LOCATION
2	TOP	7	
3	EPICS_BASE	8	
4	SUPPORT	9	
5	ENGINEER	10	

Heartbeat service

- Frequency set by SCAN rate (default to 10 sec)
- Heartbeat VAL increments when record processes
- Heartbeat UDP packet contents:
    - Magic number (for filtering)
    - Protocol version (4 currently)
    - Incarnation (boot time) and current time
    - Heartbeat value
    - Flags (currently for info port)
    - Information port number
    - 32-bit user message MSG
    - IOC name

Information Port Service

- Initialized by remote server, by making TCP connection.
- Port number can be specified or automatically assigned.
- If initialization fails, thread terminates, and sets status to “Inoperable” (status is “Operable” on success).
- Queries only allowed by IP of server heartbeats sent to.
- Record can request a reading with flag, using ITRG
- Record can suppress connections using ISUP, where connections are denied, with a flag sent indicating this.

Information Port Service

- Information contents
  - Protocol Version (4)
  - IOC type (currently vxWorks, Linux, and Darwin)
  - Total message length
  - Data
    - For each ENVxx field that is not empty, there is the variable name and its value
    - IOC type specific information
      - VxWorks: bootLine
      - Linux/Darwin: user, group, and host

Implementing Server

- Heartbeat Processing
  - Toss out if magic number is wrong.
  - Match version against supported values.
  - Find IOC entry, create if needed (if allowed).
  - If incarnation has changed (or new), boot has occurred, reset entry and read IOC information.
  - If heartbeat value is lower, toss (out of order packet).
  - Record current time as ping time, IOC's measured time, and user message value.
  - If flag bit 1 is set, can't do information read.
  - If flag bit 0 is set, try to do information read.

Implementing Server

- Failure determination
  - Failure time is determined by SCAN rate and necessary number of missing heartbeats.
  - 60 second failure time with 10 second scan rate means six missing heartbeats
  - Elapsed time is current time – ping time
- Information Reading
  - Open TCP port using value from heartbeat
  - Read stream until closed (use message size field for error checking)
  - Attach information to IOC record

BCDA server

- Allows any IOC to join
- Currently has around 100 IOCs active
- Design
  - Written in C as threaded daemon.
  - Database is autobalancing tree, uses many-reader, single-writer model, preferring the writer.
  - Clients access data over TCP port, using API.
  - Records IOC state in case of restart.
  - Records each boot for every IOC.
  - Lets client do failure determination.

BCDA clients

- CGI: http://bcda.xray.aps.anl.gov/cgi-bin/ioc_alive.cgi
- CGI XML: http://bcda.xray.aps.anl.gov/cgi-bin/alivexml.cgi
- Command line: /APSshare/bin/alivedb
- Command Line XML: /APSshare/bin/alivexml

The XML interfaces lets one use an XML parser for loading the database.

Conclusion

Development location (has HTML documentation):

https://subversion.xray.aps.anl.gov/synApps/alive/trunk/

Future Plans:

- Get it fully released soon as module
- Figure out how to give server code example
- Add notifier mechanism for running a script