Tutorial: Programming for Linux-HA

Alan Robertson
IBM Linux Technology Center
alanr@unix.sh OR alanr@us.ibm.com

Background on Linux-HA

- Linux-HA is the oldest high-availability project for Linux, with the largest associated community
- The core piece of Linux-HA is called “heartbeat” (though it does much more than heartbeat)
- Linux-HA has been in production since 1999, and is currently in use on thousands of production sites
- Linux-HA also runs on FreeBSD and Solaris, and is being ported to OpenBSD and AIX

Linux-HA Programming Tutorial Agenda

- Background and Motivation
- Structure of Heartbeat
- Resource Scripts
- The Linux-HA client API
- Application Heartbeat API
- PILS - Plugin and Interface Loading System
- Clplumbing library
- Plugin APIs:
  - STONITH / Communication / Authentication
- Consensus Cluster Membership (CCM)

Hacking Linux-HA

(Linux-HA is eminently hackable)

- Figure out what you want to do (check TODO list)
- Discuss it on the linux-ha-dev mailing list
- Understand enough of the structure around what you want to do to get an idea of what’s up
- Plugin creation is very common
- Find a component similar to the one you want
- Use it as a template
- Hack away
- Submit results to mailing list (with appropriate disclaimer)
Interesting TODO List Items

- CRM - Large cluster Cluster Resource Manager (currently underway)
- LRM - local resource manager process (also underway)
- initial configuration installation tool
- Management interface-type tool
- Complete, improve documentation
- Support OCF resource agents
- Add support for "warm failover" resources
- Data checkpointing API (underway)

Linux-HA directory structure

- cts : Cluster Testing System (Python)
- include : linux-a header files directory
- libdir : Linux Director Daemon - monitors LVS real services
- replace : library replacement functions (inet_pton, selinux, etc)
- libr
  - appslib : client side application heartbeat library
  - plib : Plug system library
  - recoverymgr : recovery manager client library
  - cplumbing : cluster plumbing library (IPC, CPU, realtime, etc)
- plugins
  - AppNotification : Recovery manager notification plugin
  - l8auth : heartbeat authentication (CRC, MD5 and SHA-0)
  - InterfaceMgr : Generic interface (implementation) manager
  - stonith : stonith plugins
- stonith : stonith library and utility

Interesting TODO List Items (cont'd)

- Add script-oriented STONITH interface/plugin
- Write STONITH plugin for NUT-supported UPSes (i.e., plugin for NUT)
- Add non-STONITH fencing
- Improve test suite
- Add an ordered messaging API to heartbeat
- Write a respawn service to restart scripts
- Canned configuration docs

Linux-HA directory structure (cont'd)

- snmp_subagent : SNMP remote management for Linux-HA cluster
doc : Linux-HA documentation
libld : GNU libdld, a system independent dlopen wrapper for libiod

- ccmand : Consensus Cluster Manager
dir : Directory for cluster resource managers
- crm : Directory for cluster resource managers
- scmt : New Cluster resource Manager implementation
- heartbeat : original heartbeat framework
- contrib
  - ifail : IP failover plugin for Linux-HA
tools
  - apphbd : application heartbeat daemon
  - recoverymgrd : Recovery Manager
  (apphbd error recovery only)
Heartbeat Structure

- Heartbeat has a multi-process architecture - for security and historical reasons
- There are currently four types of processes
  - Master Control Process
  - FIFO reader process
  - Read process
  - Write process

Master Control Process

- Sends heartbeats
- Times out peers who fail to heartbeat
- Implements our reliable multicast communications protocol
- Uses HBauth plugins to authenticate / sign packets
- Manages HA cluster resources (currently)
- Implements server side of heartbeat client API
- Starts (and restarts) client processes
- Uses the glib mainloop construct for event management, timers, etc.

FIFO reader process

- The FIFO reader process is very simple
- It reads messages from the FIFO in /var/lib/heartbeat/fifo
- It copies them to a UNIX domain socket via our IPC library
- Messages read from the FIFO are sent to the cluster by the Master Control Process
- This is so the master control process will (eventually) won’t have to deal with FIFOs
Read process

- The read processes' job is to get packets from various cluster communication media and give them to the master control process.
- There is one read process for each configured communication medium.
- This is similar to the FIFO reader, except they read from external communication media.
- Read processes start as root, but run as nobody.
- Read processes get their communication capabilities from HBcomm plugins.

Write process

- The write processes job is to send packets to various cluster communication media as directed.
- There is one write process for each configured communication medium.
- Write processes get their communication capabilities from HBcomm plugins.
- Write processes start as root, but run as nobody.

Heartbeat Interfaces

- Heartbeat currently has three types of plugins:
  - STONITH plugins
  - Communication plugins
  - Authentication plugins
- I've also thought about adding plugins for
  - Fencing policies
  - Resource Management

Heartbeat plugins
The Heartbeat resource model

► Everything that heartbeat manages on behalf of the administrator is called a resource
► A resource is an abstract entity which might represent almost anything:
  ► A service (like Apache)
  ► An IP address
  ► A piece of hardware (like a disk)
  ► A kernel resource - like a file system, or firewall rules
  ► A pseudo-resource - like a notification script

Current Resource Script Weirdness

► For the status operation only, the return code is currently IGNORED
► This is for compatibility with older Linux releases that always exited with 0.
► Instead the output is examined for the string OK or running. If found, the resource is assumed to be active - otherwise it's assumed to be inactive.
► We'll fix this when we change to OCF resource agents (which are LSB-compliant).

What's a resource script?

► The main thing heartbeat cares about regarding a resource is that it can:
  ► Start (enable) it
  ► Stop (disable) it
  ► Tell if it's running (enabled)
► This is very similar to an /etc/init.d script
  ► Heartbeat extends normal init interfaces:
  ▶ By adding optional "instance" parameters
  ▶ By adding a "monitor" operation

MailTo Resource script – Initialization

```
# MailTo resource script.
# Sends mail when we take over or give up
# resources in this resource group
# This is a pseudo-resource (but it's simple :))
# has resources syntax: MailTo::mail130mx,sh::WebServer_takover
# Source the standard resource script shell functions
#.$(HA_DIR)/shellfuncs ./etc/hd.d/shellfuncs
STATFILE=/var/lib/heartbeat/rsc/tmp/MailTo
ARGS="\$1 $2"
usage() { echo "Usage: 50 enviarhost [subject] {start|stop|status}" exit 1 }
```
MailTo Resource script – Functions

MailProgram() {
    echo -n "$Subject\nSubject:\n\n$From\n\n$Reply-To\n\n$Body\n\n$m"
    mail -s "$Subject" "$To"
}

SubjectLine() {
    case "$1" in
        ???)  echo "$2;"
        *)    echo "Resource Group:"; esac
    MailToStart() {
        Subject="" SubjectLine "$subject" "takeover in progress on $subject"
        MailProgram "$subject" "$1"
        touch $STATEFILE
    }
    MailToStop () {
        Subject="" SubjectLine "$subject" "migrating away from $subject"
        MailProgram "$subject" "$1"
        rm $STATEFILE
        esac
    }

Heartbeat Client API

- Provides basic low-level cluster services:
  - Low-level “membership” status / notification
  - Communication link status / notification
  - Cluster communication
  - Cluster topology retrieval
    - nodes
    - communication links
  - Configuration parameter retrieval
  - API notifications are event-oriented

API Client applications

- api_test - does cluster-wide ping, etc.
- ipfail - monitors “ping” nodes and moves resources when IP connectivity fails
- CCM - Consensus Cluster Membership
  - Provides consistent cluster-wide membership information
  - Has a client interface for CCM clients
  - Client interface is OCF-compliant
Planned/Potential API Clients

- **clstat** - provide cluster status information
- **CRM** - full fledged n-node cluster resource manager (Also a CCM client)
- **SNMP** support
- **SBLIM** systems management support

Heartbeat Client API – part 1

- **General message receipt callback**
  ```c
  int (*set_msg_callback)(ll_cluster_t* clusterinfo, const char* msgtype, llc_msg_callback_t cbf, /* function pointer */ void* p);
  ```
- **Node status update callback**
  ```c
  int (*set_nstatus_callback)(ll_cluster_t* clusterinfo, llc_nstatus_callback t cbf, /* function pointer */ void* p);
  ```
- **Communication status update callback**
  ```c
  int (*set_ifstatus_callback)(ll_cluster_t* clusterinfo, llc_ifstatus_callback t cbf, /* function pointer */ void* p);
  ```

Heartbeat Client API – part 2

- **Get current node status**
  ```c
  const char* (*node_status)(ll_cluster_t* clusterinfo, const char* nodename);
  ```
- **Return "type" of node**
  ```c
  const char* (*node_type)(ll_cluster_t* clusterinfo, const char* nodename);
  ```
- **Return current interface status**
  ```c
  const char* (*if_status)(ll_cluster_t* clusterinfo, const char* nodename, const char* interface);
  ```

Heartbeat Client API – part 3

- **Send message to all nodes in cluster**
  ```c
  const char* (*sendclustermsg)(ll_cluster_t* clusterinfo, struct ha_msg* message);
  ```
- **Send a message to a specific node**
  ```c
  const char* (*sendnodesmsg)(ll_cluster_t* clusterinfo, struct ha_msg* message, const char* nodename);
  ```

UKUUG Winter 2003 Tutorial - Programming for Linux-HA -- Page 7
Heartbeat Client API – part 4
receiving messages

- Activate callbacks for incoming message, ignore msgs w/o callback
  - `const char* (*rcvmsg)(Il_cluster_t* clusterinfo, int blocking);`
- Return next message not intercepted by a callback
  - `struct ha_msg* (*readmsg)(Il_cluster_t* clusterinfo, int blocking);`
- Associate signal with message receipt
  - `int (*getmsgsignal)(Il_cluster_t* clusterinfo, int signal);`
- Return file descriptor for use with poll(2), select(2) etc
  - `int (*inotify)(Il_cluster_t* clusterinfo);`
- Return IPC_Channel for use with mainloop
  - `IPC_Channel* (*pechann)(Il_cluster_t* clusterinfo);`

Application Heartbeat Service

- “Watchdog” API for HA-aware applications
  - Register with service
  - Declare promised “heartbeat” time
  - Heartbeat periodically
  - Unregister
  - Notification provided for applications which die, or don’t heartbeat
  - Notification is sent to plugins

Heartbeat Client API – part 5
miscellaneous calls

- Return value of given configuration parameter
  - `char* (*get_parameter)(Il_cluster_t* clusterinfo, const char* parameter_name);`
- Return out node_id
  - `const char* (*get_mynodeid)(Il_cluster_t* clusterinfo);`
- Return current heartbeat keepalive interval
  - `long (*get_keepalive)(Il_cluster_t* clusterinfo);`
- Return current heartbeat deadline interval
  - `long (*get_deadline)(Il_cluster_t* clusterinfo);`
- Return current heartbeat cluster resource ownership
  - `const char* (*get_ownership)(Il_cluster_t* clusterinfo);`
- Return last error message
  - `const char* (*errmsg)(Il_cluster_t* clusterinfo);`

Application Heartbeat API

- `int apphb_register(const char* appname, const char* appinstance);`
- `int apphb_unregister(void);`
- `int apphb_setinterval(unsigned long hbms);`
- `int apphb_setwarn(unsigned long hbms);`
- `int apphb_hb(void);`
PILS – Plugin and Interface Loading System

- PILS provides uniform plugin services for Linux-HA
- PILS provides for function imports as well as exports
- Described in 2002 OLS paper
- Common Plugin Imported functions:
  - register_plugin() / unregister_plugin()
  - register_interface()
  - load_plugin()
  - log()
  - alloc() / free()

CLPLUMBING – Cluster Infrastructure Layer

- lib/clplumbing: general utility functions
  - cl_log – logging functions
  - cl_signal – signal handling wrappers
  - ipc – nice non-blocking IPC layer
  - longclock_t – 64-bit clock_t replacement
  - realtime – provide basic real time APIs
  - proclock – track child processes
  - cl_poll – poll(2) replacement with better real-time behavior
  - uid_s – manage privileges
  - base64 – converts to/from base 64
  - Gsource – integrates IPC with glib mainloop
  - Gmain_timeout – longclock_t based mainloop timeouts

Our use of Glib

- We use the “Glib” library (from GNOME project)
- Glib is not a graphics library
- Glib is not the same as the GNU GlibC
- Our usage of Glib:
  - Basic data structures (lists, hash tables)
  - Mainloop scheduler and dispatcher
  - It has worked out quite nicely for us

Heartbeat IPC layer

- Higher-level non-blocking sockets-like communication layer
- Oriented to building non-blocking applications without requiring threading
- Interface is generic for any kind of IPC implementation
- Includes authentication methods
- Could be fit on top of most communication layers
- Requirement is that it has corresponding file descriptors for poll/select, etc.
IPC layer API paradigm

- Three basic kinds of IPC objects:
  - IPC_WaitConnection – server side: wait for client connection
  - IPC Channel – symmetric communication
  - IPC_Message – the unit of exchange between endpoints

IPC_WaitConnection

- This is a “server-side” object only
- Created by ipc_wait_conn_constructor()
- Basic Operations:
  - accept_connection() – accept client connection
    - Returns an IPC_Channel object
  - get_select_fd() – returns file descriptor for poll/select
  - destroy() – destroys the IPC_WaitConnection

IPC_Channel Object

- Fully Buffered IPC Channel
- Created by IPC_WaitConnection::accept_connection() or ipc_channel_pair()
- Basic Operations:
  - destroy()
  - initiate_connection()
  - verify_auth() – verify credentials against sets
  - send() – send IPC_Message
  - recv() – receive IPC_Message

IPC_Channel – continued

- waitn() – suspend waiting for input message
- waitout() – suspend waiting for output to complete
- is_message_pending()
- is_sending_blocked()
- resume_lq() – send output, read input
- get_send_select_fd()
- get_recv_select_fd()
- set_send_qlen() – set maximum output message queue length
- set_recv_qlen() – set maximum input message queue length
IPC_Message object

- msg_len field - length of message in bytes
- msg_body field - Pointer to message body
- msg_done() - function to call to free message
- msg_private field - pointer to whatever
- msg_ch field - pointer to associated IPC_Channel

You construct these messages for output
System constructs them for input

STONITH Plugin API

- STONITH == Shoot The Other Node In The Head
- STONITH software is library-level user code
- STONITH device drivers are loaded as plugins

STONITH Plugin Exports

- new() - constructs STONITH object of given type
- destroy() - destroys STONITH object
- set_config_file() - configures this STONITH object
- set_config_info() - configures this STONITH object
- getinfo() - return information about this object
- reset_req() - perform reset-type operation
- hostlist() - return list of hosts this object controls
- free_hostlist() - frees return from hostlist()

Communication Plugin API

- All heartbeat communication is done by plugins
- Currently supported communications media:
  - Serial port
  - UDP broadcast
  - UDP multicast
  - UDP unicast
**HBcomm** **Imported** functions

- These functions are available for HBcomm plugins
  - `ParamValue()` - returns value of configuration parameter
  - `RegisterNewMedium()` - call to register ourselves with system
  - `devlock()` - called to lock device (e.g., tty device)
  - `devunlock()` - called to unlock device
  - `StrToBaud()` - convert a string to a baud rate #define
  - `RegisterCleanup()` - register our shutdown/cleanup routine
  - `CheckForEvents()` - call periodically to process signals, etc.

**Authentication Plugin API**

- All heartbeat communications is authenticated by digital signatures
- All cluster members have the same signing key
- Each node signs packets with one key
- Each node can validate packets with multiple keys
- This allows for graceful key changes
- Currently supported methods:
  - `SHA1, MD5 and CRC (CRC is insecure!)`

**HBcomm** **Exported** functions

- These functions are exported by HBcomm plugins
  - `new()` - parses single configuration token
  - `parse()` - parses complex configuration line (new() or parse(), but not both must be NULL)
  - `open()` - called before forking to open for reading/writing
  - `close()` - duh...?
  - `read()` - return a packet from the communication medium
  - `write()` - send a packet to the communication medium
  - `mtype()` - return short media name
  - `desc()` - return longer medium description
  - `isping()` - returns TRUE if this is a ping medium

**HBauth exported functions**

- `auth()` returns digital signature on a packet
- `needskey()` returns TRUE if the authentication method needs to be supplied a key.
- **NOTE:** there are no HBauth imported functions.
**What’s CCM?**

- CCM == Consensus Cluster Membership
- When your cluster has communication problems, it breaks down into *cluster partitions*.
- This is sometimes called “split brain”
- CCM defines/determines membership in these partitions
- Each node is in exactly one partition at a time
- All nodes in a given cluster partition have the same membership at a given time
- Robust with respect to communication failures

**CCM API(s)**

- Current CCM implementation conforms to draft 0.4 of the OCF membership API specification
- Will convert to the current OCF membership API
- Will also provide an SAF API interface
- *CCM services are naturally event-oriented*

**References**

- [http://linux-ha.org/](http://linux-ha.org/)
- [http://linux-ha.org/HATodo.html](http://linux-ha.org/HATodo.html)
- [http://linux-ha.org/doc/](http://linux-ha.org/doc/)
- [http://linux-ha.org/heartbeat/heartbeat_api.html](http://linux-ha.org/heartbeat/heartbeat_api.html)
- [http://linux-ha.org/heartbeat/pils.pdf](http://linux-ha.org/heartbeat/pils.pdf)
- [http://linux-ha.org/heartbeat/PLS.pdf](http://linux-ha.org/heartbeat/PLS.pdf)

**Legal Statements**

- IBM is a trademark of International Business Machines Corporation.
- Linux is a registered trademark of Linus Torvalds.
- Other company, product, and service names may be trademarks or service marks of others.
- This work represents the views of the author and does not necessarily reflect the views of the IBM Corporation.