Venus Data Structures

Lily Mummert v1.0 5/21/97

This is a short summary of the essential data structures used in Venus.

1 Introduction

The data structures used in Venus fall into the following categories:

- File cache
- Volumes
- Communication - servers, MRPC, VSG's
- Hoard database
- Users
- Threads
- Local repair
- Data collection

A characteristic feature is that some data structures function as an umbrella for a table of others. We mean that a single instance of them is present. Others appears as tables, trees, linked lists etc.

Some data structures are persistent and stored in RVM. On the whole modification of such structures involves RVM transactions. Sometimes components of persistent structures are labelled as transients. Such transients are stored in RVM but have no persistent meaning. These are generally marked in the source with `/* T */`.

2 File Cache

2.1 FSDB – the file system database (persisent)

This is an umbrella structure.

- fixed number of slots, blocks
- Contents:
  - links to FSOs:
    - FSO hash table
    - priority queue (btree)
    - delete queue
    - open for write queue
  - statistics
  - block, file counts
  - priority weights

2.2 FSOBJ – a slot in the cache (persisent, some transient fields)

The fso class contains most of the information about a file system object. It has pointers to the volume information, part of which is copied to avoid lookups. There are pointers to the data, which can be a symlink, file or directory object. Data structures of type binding are used to store hoard and cml information.

Fso objects keep track of readers and writers of the object.

- fid
- component name
- state (normal, runt, dying)
- status block (Venus stat, a condensed version of ViceStatus, see vcrcommon) vnode type in here, and size, mtime, etc.
- access control information
- flags (class FsoFlags)
  - fake, fetching, replaceable, etc.
  - for volume types (copied) e.g. backup, replicated, etc.
- mount state (normal, mount point, mount root) links to root, mount point in latter cases
- data
  - symlinks stored directly in RVM
  - files stored as container files (CacheFile)
  - directories stored in RVM written to container files upon open
- counts (readers, writers, openers, etc.)
- links to
  - hash table (fsdb)
  - volume
  - priority queue
  - delete queue
  - open-for-write queue
  - parent fso
  - list of child fsos
  - HDB bindings
  - MLE bindings
- statistics

2.3 CacheFile – cache container file

The fso objects are related one-one to cache files. Methods exported by cache files are simple on these are things like validate, move, copy, remove, truncate, reset. It appears that historically cachefiles could be accessed through inode calls. Currently cache files and fsobjs are statically linked by embedding.

- statically bound to fsobj
- container files used in several ways
  - to store a plain file
  - to store a UNIX format directory as a plain file (NB: this format is system independent now!)

3 Useful data structures

binding – used for grid-like data structures (e.g. fso x cmlent) pointers to binder, bindee, next/prev in both lists.

The `util` library is full of other useful structures.

4 Volumes

Volumes are recorded using an umbrella data structure called the volume database (VDB), defined in venusvol.h.

- volume database. (persistent)
- contains volents (linked),
- contains mle
- resource limits on MLEs

Contained in the volume database are above all entries for individual volumes: volents. Volents are stored in persistent storage. Volents are described in venusvol.h as well and contain the following important fields:

- volume ID (of course)
- state (hoarding, emulating, logging (write disconnected),etc.)
- flags (state transition pending, reintegrating, repairing, weakly connected, etc.) counts (readers, writers, etc. for synchronization)
- range pre-allocated fids
- version vector, callback status
- list of read/write volume ids (if replicated)
- replicated volume id (if read-write)
- CML
- link to VSG entry (vsgent)

- list of FSOs
- list of COP2 pending (cop2ent)
- list of entries requiring resolution (resent)
- statistics (vmon, etc.)

XXX What are observers? What are pre-allocated fids?

A volume carries a client modification log. The umbrella data structure is the ClientModifyLog, which contains entries of type cmlent. These are all persistent. A structure of type ClientModifyLog has an owner, who is responsible for writing all the entries of the log. XXX What is an mle versus a cmle

- list of cmlents
- ownership (only one user may own)
- statistics

An entry in the CML is a higly technical structure reflecting the filesystem updates. It contains opcodes and a union (u) of data containing the fid and update details. All cmlents are held in RVM.

- opcode, operands
- flags (frozen, local repair, etc.)
- links to
  - CML
  - fsobjs
  - predecessor, successor

A very small structure holds the data associated to a cop2 event. This data is transient. cop2ent – entry for pending COP2 events (transient)

- link to volume
- store ID
- update set

Another small structure holds data associated with fids needing resolution.

- resent – entry for fids needing resolution (transient)
- fid, result

5 Comms

The file comm.h contains definitions for the datastructures for communication. Key structures map the id's of replicated volumes to servers, and volume storage groups to hosts. Currently this is hacked into venusvol.cc; it would be better to make an rpc for this information.

First there is an umbrella structure:

- VSGDB – VSG database (persistent)
- contains vsgents

Entries are:

- vsgent – VSG entry (persistent)
- maps VSG address to list of hosts

The transient structures holding connection information are stored in connent structures. These document single connections between clients and servers.

- connent – connection entry (transient)
- host
- uid
- RPC2 conn id
- flags (authenticated connection? inuse? etc.)

Servers are held in srvent's.

- srvent – server entry (transient)
- name
- host
- RPC2 conn id (for callback connection) tests for up/down rolled on this field
- bandwidth estimate
- flags (binding? need to probe?, etc.)

The data describing multirpc connections is held in mgrpent's:

- mgrpent – multirpc connection entry (transient)
- vsgaddr (should be link to vsgent?)
- uid
- hosts
- flags (authenticated? in use? etc.)
- RepOpCommCtxt
  - RepOpCommCtxt – replicated operation communication context (transient)
  - number of servers
  - list of conn ids
  - list of hosts (actually participating in RPC)
  - list of return codes
  - primary host
  - flags (per host)

The primary host should be used for load balancing. Currently the algorithm is not satisfactory. The primary host is also (?) held in the fso status block.

6 Hoard Database

The main umbrella structure is the hdb.

- hdb (persistent)
- HDB entries
- priority queue
- resource limits

The entries in the hdb are hdbent:

- hdbent - a single entry (persistent)
- object represented as vid, path
- uid
- priorty
- flags (c+, d+?)
- link to name context

Name contexts are used for renames:

- name context – for directory expansion (persistent)
- uid
- priority
- flags (inuse? expanded? etc.)
- links to
  - expanded children
  - parent (back pointer to expander)
  - HDB priority queue

7 Users

- userent – user specific information (transient)
- uid
- tokens
- advice monitor methods go through here

8 Threads

Threads in Venus come in about a dozen of different types. Basic threads handlnig vfs requests are object of type vproc; see vproc.h. These classes export a very important interface encapsulating the type of messages and operation dispatched to threads during operation. Among the exported operations are the vfs operations, which are spelled out in vproc_vfscalls.cc.

Pioctl calls are handled by routines defined in vproc_pioctl.cc. This file is effectively one large case statement to distinguish among opcodes.

vproc_pathname.cc handles the lookup of names in directories and fills in a vnode as output. It also contains a routine translating fids to full pathnames.

Typically the routines in these files call methods of the FSDB to find the appropiate objects and manipulate their data.

Operations such as resolution and re-integration are handled by special daemon threads.

- vproc – basic thread type. handles requests in VFS interface (transient)
- type (worker, mariner, callback, resolver, reintegrator, daemons)
- LWP id
- LWP priority
- message ID (sequence number)
- flags (idle, prefetch, etc.)
- name
- function
- vpid (venus thread id)
- uarea – things the thread needs to carry with it
  - return code
  - user identifiers
  - priority
  - link to volume
  - vfs operation
  - process/group id of calling process
  - counts to guard against infinite retry loops
- data collection

A derived class from the vprocs are the worker threads (worker.h). These threads handle the messages delivered to Venus. Their interface is defined in worker.cc which describes the methods for handling messages. This file also contains the operations for mounting /coda. A message queue is maintained. Messages are dispatched to a pool of threads.

- worker – generic worker thread, derived from vproc
- opcode,
- message

daemon threads for each subsystem (volume, comm, rvm, etc.)

9 Messages

Messages are the data sent to Venus by the kernel. The structures are transient. Definitions can be found in worker.h.

- msgent – requests from kernel (worker.h)
- message buffer

10 Local repair

On the whole the feeling is that local repair stuff needs to be redone. At the moment the following code is present:

- LRDB (persistent)
- list of local-global fid mappings
- list of local subtree roots
- list of fsobjs, volumes, cmls involved in local repair

- lgment – local-global map entry (persistent)
- local fid
- global fid

- rfment – root fid map entry (persistent)
- name
- fake, global, local root fids
- parent, child fids

Numerous other structures are present, probably most can be simplified. Much of this is transient.

11 Recoverable data

The main structure containing pointers to RVM tables and other data is held in the structure RecovVenusGlobals. The data structure is straightforward and defined in venusrecov.h. In the venusrecov.cc the methods are defined. These contains numerous calls to rvm initialization and recovery stuff.

- structure RecovVenusGlobals
- magic numbers, last init time, etc.
- root volume name
- pointers to
  - FSDB

  - VDB
  - VSGDB
  - HDB
  - LRDB
  - VMSE (vmon session data)
  - VCBDB
  - heap address (base of recoverable heap)
  - heap length

Transient fields in recoverable structures are marked with /*T*/.

12 Data Collection

- VCBDB, vcbdent – data collection for volume callbacks,
- uses mond

- vsr – volume session record. cache statistics, etc.
- uses mond

- rwsent – read-write sharing statistics
- uses mond