Changelogs and Feeds

Nathan Rutman

1/30/08

1 Use cases

1.1 File post-processing

A post-processing operation is to be started whenever a new file is created in /srv/cam1, /srv/cam2, or /srv/cam4. The post-processing operation needs the filename, and should be started when the new file is first closed.

1. Fileset 'towatch' is created using fileset API, below
2. Directories are added to 'towatch' using fileset API
3. Feed is created to watch CREATE and OPEN/CLOSE events on this fileset
4. The feed FIFO is opened by the consumer
5. The consumer loops on feed entries with a blocking read on the feed FIFO. The consumer checks to see when newly created files are first closed.

```
llapi_fileset_new("towatch");
llapi_fileset_add("towatch", "/srv/cam1");
llapi_fileset_add("towatch", "/srv/cam2");
llapi_fileset_add("towatch", "/srv/cam4");
struct feed_policy policy={fp_filtermask=FF_CREATE|FF_OPEN};
llapi_feed_new("towatch", *policy);
fd=open($MNT/.lustre/feed/towatch, O_RDONLY);
loop {
    read(fd, struct feed_entry *entrybuf);
    if entrybuf->fe_type = create then add fe_data.fid to createdlist;
    else if entrybuf->fe_type = close and fe_data.fid is on createdlist then
        postprocess(llapi_fid2path(fe_data.fid));
}
```

2 User APIs

2.1 Fileset API

Start a new fileset definition:

```
int llapi_fileset_new(char *fileset);
```

Add a file or directory to a fileset:

```
int llapi_fileset_add(char *fileset, char *filename);
```

Directories added to a fileset refer to the entire subtree. Moving a file out of a subtree removes it from the fileset. A file or directory explicitly added to a fileset retains its membership if renamed.

Remove a file or directory from a fileset:

```
int llapi_fileset_remove(char *fileset, char *filename);
```

A child file or directory may be removed from a fileset that includes the parent. This has the effect of pruning a subtree out of the fileset tree.

Destroy the fileset :

```
int llapi_fileset_destroy(char *fileset);
```

A fileset cannot be destroyed while in use by a feed, mount, etc. (EBUSY).

Retrieve status or other info about a fileset:

```
int llapi_fileset_getinfo(char *fileset, struct fileset_info *info)
```

2.2 Feed API

Feeds provide userspace access to a server changelog. A single user process (consumer) may access a feed. Feeds are transactional and persistent; feed entries are guaranteed to be replayable in the event of a server restart, from the point where the consumer last indicated completion.

2.2.1 Feed content

Feed entries will be packed binary data, with the form

```
struct feed_entry {
    _u32 fe_len;       total record length
    _u32 fe_type;      transaction type
    _u64 fe_seq;       local feed sequence number
    _u64 fe_cookie;    synchronization cookie (for distributed events)
    _u64 fe_time;      event time, server-local
    _u32 fe_result;    return code (0=success)
    void *fe_data;     transaction type-specific struct
```

Transaction types:

```
enum {t_create, t_unlink, t_open, t_close, t_read, t_write, t_attrib, t_rename, t_link
```

Transaction type-specific struct contains event-specific data. (Ideally this will contain enough data for a userspace filesystem replicator; in many cases a MDT_REINT structure would be sufficient.) For example:

```
struct feed_entry_open {
    ll_fid fid;          (see lustre_idl.h)
    ll_fid parent_fid;
    nid_t clientnid;
    _u32 fsuid;
    _u32 fsgid;
    _u32 cap;
    _u32 flags;
    _u32 mode;
    _u32 filename_len;
    char *filename;
```

Feed content examples (expressed in human-readable form, produced by a Sun provided feed consumer demo utility):

```
logid=1 cookie=0 type=OPEN rc=0 name=/etc/passwd fid=23a87346:003d source=cli1@tcp0
logid=2 cookie=0 type=UNLINK rc=-EACCESS fid=23a87346:003d source=cli2@tcp0 uid=joe
```

2.2.2 Feed setup

A feed is created (and available) on a single Lustre client as a FIFO file. New feeds are defined through lfs or a direct call to the liblustre c library (llapi).

```
int llapi_feed_new(char *fileset, struct feed_policy *policy)
```

starts a new audit feed. <fileset> is a previously defined fileset or a server name. Filesets are defined via the Fileset API. Special user permissions are required to start an audit log; else EPERM is returned. The new feed is created as a FIFO at $MNT/.lustre/feed/<feedname>, where <feedname> is <fileset>[_xx], with increasing numerical xx if the name already exists.

<policy> is a structure containing

```
struct feed_policy {
    _u32 fp_filtermask;
    _u32 fp_entry_timeout;
    _u32 fp_abort_timeout;
    _u32 fp_abort_size;
    int  fp_flags;
```

<filtermask> is a bitwise event mask:

mask bit	description	masked types
FF_CREATE	new file creation	t_create
FF_WRITE	file modify/append	t_write
FF_READ	file read	t_read
FF_OPEN	open/close	t_open, t_close
FF_ATTRIB	file attribute / EA change	t_attrib
FF_DELETE	file removal	t_unlink
FF_LINK	soft/hard link	t_link
FF_RENAME	rename	t_rename
FF_FILE	all of the above (shortcut)	t_create, t_unlink, t_open, t_close, t_read, t_write, t_attrib, t_rename, t_link
FF_ADMIN	administrative event	t_admin
FF_ERR	report failed requests also	err & (t_create, t_unlink, t_open, t_close, t_read, t_write, t_attrib, t_rename, t_link, t_admin)
FF_REPLICATE	all events related to fileset replication	t_create, t_unlink, t_write, t_attrib, t_rename, t_link

Retention policies (see Feed I/O below) may include:

- entry_timeout - automatically cancel each feed record after X seconds (0=off, default=0).
- abort_timeout - abort recording and destroy the feed after X second timeout (0=off, default=0).
- abort_size - abort recording after the number of unconsumed records exceeds a maximum (0=off, default=1000).
- flags
  - FG_RATELIMIT - don't report multiple consecutive similar entries.

The feed setup remains persistent across reboots (in e.g. a feed database), until it is explicitly destroyed:

```
int llapi_feed_destroy(char *feedname)
```

Feed setup info and status is available for retrieval from an existing feed:

```
int llapi_feed_getinfo(char *feedname, struct feed_policy *policy, struct feed_status
```

Two functions for converting FID to filename or full path name are also provided. Full path name is at the filesystem's disgression for hard-linked files.

```
int llapi_fid2file(ll_fid fid, char *filename);
int llapi_fid2path(ll_fid fid, char *pathname);
```

2.2.3 Feed I/O

The feed output data stream is presented as a FIFO file (see mkfifo) under $MNT/.lustre/feed/<feedname>. A feed may be open(2)ed by only a single reader at a time. Feed entries are retrieved using read(2) on the file. Reads will block until new entries are available, or poll(2) or select(2) may be used.

Feed entries are removed from the feed according to the following policy:

The file descriptor will make available only full feed entry records. With each read(2) (or close(2)) of the fd, all of the entries from the *previous* read(2) are marked as consumed. For example: read1 reads fe_seq 1-5, nothing is marked consumed; read2 reads fe_seq 6-7, seq 1-5 are marked consumed; close1, seq 6-7 are marked consumed.

Upon recovery, we restart the feed from the first unconsumed entry. The feed consumers are responsible for skipping any replayed feed entries they may have already processed (identified by repeated fe_seq).

2.2.4 Feed entry ordering

For consumers combining the data from multiple feeds (e.g. database), partial ordering of operations is required. Since clients synchronize the epoch among servers for sequential transactions, using the epoch is sufficient to order "before relations", while unrelated transactions may be reported in a random order. Distributed transactions are linked via a common cookie contained in all affected changelogs and reported in all affected feeds. Consumers are reponsible for linking/ignoring distributed transaction entries internally.