OVIS Logging

Overview

- Motivation
  - Currently some LDMSD subsystems cannot log messages.
- Introduce the OVIS log library
- LDMSD's log configuration
  - What's new?
    - Ability to set any combination of log levels.
    - Ability to set the log level for a subsystem.
- OVIS log library APIs

OVIS Log Library Features

- Logging a message should not interfere with application run-time.
- Support logging messages to Syslog.
- Backward-compatible with the current LDMSD implementation and configuration.
    - Does not require refactoring existing log messages
    - Supports current log level configuration commands, e.g. loglevel

OVIS Logging Library Features (cont.)

- LDMSD, underlying libraries, and plugins can access the APIs directly,
  - Enable us to support logging in authentication, underlying transports, and plugins
  - Specifically, the log function pointer is deprecated.
- Log configuration can be specified per software “subsystem”, e.g.
  - Transport plugins, authentication methods, and sampler plugins, are “subsystems”.
- Users can specify a log level combination per LDMSD subsystem.
  - During troubleshooting, users can specify increased logging for only the subsystem of interest.
- Users can specify combinations of log levels, e.g.
  - Instead of ERROR and higher severity, users can say ERROR, DEBUG.

LDMSD's Log Configuration

Log Levels

- Historically, LDMSD log levels were ordered from low to high.
  - DEBUG
  - INFO
  - WARNING
  - ERROR
  - CRITICAL, ALWAYS
  - QUIET
- QUIET can be set to disable all log messages.
- Selecting a log level would include all messages of equal or higher severity.
- The new log API allows users to select any combination of log levels.

LDMSD Pre-defined Infrastructure Subsystems

LDMSD Infrastructure Subsystems	**Descriptions**
config	LDMSD configuration error messages
xprt	Transport infrastructure
stream	Stream error messages, e.g., failed to publish stream data
sampler	Common sampler infrastructure
store	Common storage infrastructure
producer	Producer infrastructure
updater	Updater infrastructure
…	

Plugins

- A plugin is a loadable library, e.g. samplers, transports, etc…

Example Plugin Components	Descriptions
sampler.meminfo	Messages for the meminfo
auth.munge	Messages for the munge authentication plugin
store.csv	Messages for the CSV storage plugin
xprt.sock	Messages for the socket transport plugin
…	

Set Default log levels

- Command-line
  - `ldmsd -x sock:411 -v INFO                # INFO & above`
  - `ldmsd -x sock:411 -v INFO,CRITICAL# INFO & CRITICAL`
  - `ldmsd -x sock:411 -v INFO,               # Only INFO`
  - `ldmsd -x sock:411 -v QUIET               # Disable all messages`

Setting log levels

- All subsystems
  - `loglevel level=INFO                   # INFO & above`
  - `loglevel level=INFO,CRITICAL             # INFO & CRITICAL`
  - `loglevel level=INFO,                     # Only INFO`
  - `loglevel level=QUIET                     # Disable all messages`
- Subsystem specific log levels
  - `loglevel` **`subsys=sampler.meminfo`** `level=INFO,CRITICAL`
    - Set log level for only the meminfo plugin
  - `loglevel` **`subsys=sampler.*`** `level=INFO,`
    - Set log level for all sampler plugins

Listing existing subsystems

- log_list is a new configuration command that lists the available subsystems.

```
ldmsd_controller> log_list
default    ERROR,DEBUG   The default log
xprt       ERROR,DEBUG   LDMS transport infrastructure
. . .
sampler.meminfo ERROR   The meminfo sampler
. . .
```

Q/A?

Next …More on Development

OVIS Log API

OVIS Log API

-
  ```
  ovis_log_t ovis_log_register(
              const char *subsys_name,
              const char *desc)
  ```
- `void ovis_log_destroy(ovis_log_t log)`

-
  ```
  int ovis_log(ovis_log_t log, int level,
                              const char *fmt, …)
  ```

-
  ```
  int ovis_log(ovis_log_t log, int level,
                              const char *fmt, …)
  ```
- `int ovis_log_open(const char *path)`
- `int ovis_log_rotate(const char *path)`

-
  ```
  int ovis_log_set_level(const char *subsys_name,
                                   int level)
  ```
- `int ovis_log_get_level(const char *subsys_name)`
- `char *ovis_log_list()`

Create a new LDMSD log subsystem

```
ovis_log_t
ovis_log_create(const char *subsys_name, const char *desc)
```

- Create a subsystem.
- subsys_name is the LDMSD logging subsystem name string, e.g., sampler.meminfo'.
- desc is a string describing the subsystem.
- errno is set on errors.
  - EEXIST means subsystem subsys_name already exists.
  - ENOMEM means 'out of memory'.

Delete the log for a subsystem

```
void ovis_log_destroy(ovis_log_t log)
```

- Call the API when the subsystem log will not be used anymore.
- log will be removed from the collection and freed.

Message Logging APIs

```
int ovis_log(ovis_log_t log, int level,
             const char *fmt, …)

int ovis_vlog(ovis_log_t log, int level,
              va_list ap)
```

- log is the log for the subsystem the message belongs to.
- If log is NULL, the default will be used.
- level is one of the DEBUG, INFO, WARNING, ERROR, CRITICAL, and ALWAYS.

Setting the Log Level of a subsystem

```
int ovis_log_set_level(const char *subsys_name,
                       int level)
```

- Set the log level of subsystem subsys_name.
- level is the bitwise-or of the log levels to be enabled.
- Return 0 on success. Otherwise, errno is returned.
    - ENOENT means that subsys_name does not exist.
    - EINVAL means that level is invalid.

Getting the Log Level of a subsystem

```
int ovis_log_get_level(const char *subsys_name)
```

- Return the log level of subsystem subsys_name.
- Return a negative errno on errors.
  - -ENOENT means that subsystem subsys_name does not exist.

Open a log file or Use Syslog

```
int ovis_log_open(const char *path)
```

- Opens a log file at path or tell libovis_log to use Syslog.
- If path is “syslog”, libovis_log sends the log messages to Syslog.

Reopen a Log File API

```
int ovis_log_rotate(const char *new_path)
```

- Rotate the log file
  - Rename the current file to <path>.<timestamp>.
  - Close the file.
  - Open the file at new_path. If new_path is NULL, it will open the file at <path>.
- If messages are going to Syslog, this is a no-op.

List the available subsystems

```
char *ovis_log_list()
```

- Return a JSON-formatted string of the available subsystems.
- It contains the names, descriptions, and log levels of the subsystems.

Log Levels

Log Levels	Variables	Values
DEBUG	OVIS_LDEBUG	0x01
INFO	OVIS_LINFO	0x02
WARNING	OVIS_LWARNING	0x04
ERROR	OVIS_LERROR	0x08
CRITICAL	OVIS_LCRITICAL	0x10
ALWAYS	OVIS_LALWAYS	0x20

- We assign a bit to a log level so that applications can use the bitwise-or to create a combination of log levels.
- For example, set_log_level(CRITICAL | INFO)

Develop a sampler plugin

sampler_foo.c

```
#define SAMP “sampler_foo”
static ovis_log_t samp_foo_subsys;
#define sampler_foo_log(level, fmt, …) do { \
                ovis_log(samp_foo_subsys, level, fmt, ## __VA_ARGS__); \
} while (0)
…
static int sample(struct ldmsd_sampler *self)
{
            …
//          ovis_log(samp_foo_subsys, OVIS_LINFO, “Sampling new data.\n”);
            sampler_foo_log(OVIS_LINFO, “Sampling new data\n”);
            …
}
…
struct ldmsd_plugin *get_plugin()
{
            samp_foo_subsys = ovis_log_create(“sampler.” ## SAMP, “Messages from “ ## SAMP);
            …
}
static void term(void)
{
            …
            ovis_log_destroy(samp_foo_subsys)
}
```

Thank you!

Set Component-specific Log Level (cont.)

good_ldmsd.conf

```
load comp=sampler.meminfo
loglevel subsys=sampler.meminfo
level=INFO
config name=meminfo…
start name=meminfo…
```

bad_ldmsd.conf

ERROR

```
loglevel subsys=sampler.meminfo
level=INFO
```

If we're going to call 'meminfo' a component, then we need to make the loglevel parameter name 'component'

```
load name=meminfo
config name=meminfo…
start name=meminfo…
```

Change log levels

- Enable additional log levels (+)
- Not sure how I feel about this…it seems a bit cute. I would rather the syntax be declarative and context independent, i.e. it doesn't depend on the current state.
  - Change the default

    ```
    ldmsd_controller> loglevel level=+INFO
    ```
  - Change a particular component level

    ```
    ldmsd_controller> loglevel name=config level=+INFO,DEBUG
    ```
- Disable a log level (-)
  - Change the default

    ```
    ldmsd_controller> loglevel level=-INFO
    ```
  - Change a particular component level

    ```
    ldmsd_controller> loglevel name=config level=-INFO,DEBUG
    ```

- + and - can be used in config files

Set log levels Syntax

This won’t be compatible with the existing configuration. Just make INFO mean INFO and above. If the comma is present, it implies only that level

Syntax	Descriptions	Examples	Enabled log levels
(level)	[level] & above	ERROR	ERROR,CRITICAL
>=(level)		>=INFO	INFO,ERROR,CRITICAL
(level),(level)	Specific combination of log levels	INFO,CRITICAL	INFO,CRITICAL
		DEBUG,INFO	DEBUG,INFO
=(level)	A single log level	=ERROR	ERROR
		=INFO	INFO

Enable/disable log levels

```
#define ovis_loglevel_enable(logger, level)
#define ovis_loglevel_disable(logger, level)
```

- Call ovis_loglevel_enable() to **enable** level of logger.
- Call ovis_loglevel_disable() to **disable** level of logger.
- logger may be the bitwise-or of multiple libovis_log log levels.

This is the same as set, we should remove it.

Data Structure

```
This should probably be hidden from users

struct ovis_logger {
    const char *name;
    const char *desc;
    uint8_t level; /* Log level bit mask of the component */
    struct rbn rbn;
}
```

- ovis_logger is an object representing an LDMSD logging component.

sampler_base logger new API

```
struct ovis_logger *
base_logger_new(const char *name)
{
    tmp = “sampler.” + name;
    desc = “Log messages from “ + “name”;
    return ovis_logger_new(tmp, desc);
}
```

Initialize the logging process

```
we should hide all this if we can

int ovis_log_init(const char *name)
```

- Applications call `ovis_log_init()` if they want to have a dedicated thread to write to the log file.
- ovis_log_init() calls libovis_ev APIs to create a worker and define the logging event_type.
- name is the program name.

Open a log file

```
int ovis_log_open(const char *path)
```

- `ovis_log_open()` opens a log file at path.
- If path is “syslog”, libovis_log sends the log messages to Syslog.

sampler_base's subsys create API

```
ovis_log_t base_subsys_create(const char *plugin_name)
```

- Why not just have ovis_log_subsys_create(). What else does base_subsys_create do?
-
- Sampler plugins call the API to create its subsys object in the `get_plugin` function.
- name is the plugin name string.
- Return a logging subsys object
- errno is set on errors.
- The API is to control the subsys name.
  - Tentatively, the subsys name is "sampler.name".

Create a new LDMSD logging subsystem

```
ovis_log_t ovis_log_subsys_create(const char *name, const char *desc)
{
     ovis_log_t subsys = find_subsys(name);
     if (subsys) {
          errno = EEXIST;
          return NULL;
     }

     subsys->desc = strdup(desc);
     return subsys;
}
```

sampler_base’s subsys destroy API

```
Same here

void base_subsys_destroy(ovis_log_t subsys)
```

- Sampler plugins call the API to destroy its sub system in the `term` function.