Debugging and Logging

Debugging and logging are both inseparable aspects of any software system during its development as well as in the deployment stage. These tools greatly assist the developer & support teams to quickly find a solution.

- Debugging tools and methodologies identifies what went wrong thus enabling developer to fix the issue quickly.
- Logging enables to maintain a history of how a software or the piece of code behave in different environments (whether it is in development or in field deployment). It helps in catching nagging field issues which are otherwise impossible to reproduce in a simulated environment. Also the other advantage it provides by generating a solid data set for any analytics process.

Debugging

Breakpad is a library and tool suite that record crashes in compact “minidump” files, send them back to your server, and produce C and C++ stack traces from these minidumps. Breakpad can also write minidumps on request for programs that have not crashed.

Refer Debugging using Google Breakpad for detailed information on Breakpad Capabilities, how to set up Breakpad, creating and analysing minidump, etc.

Breakpad C Wrapper

- A wrapper to integrate Google Breakpad with RDK components.
- Purpose is to avoid static linking of the library (libbreakpad_client.a) across different modules
- Includes the static library (libbreakpad_client.a) so that we can use dynamic library (libbreakpadwrapper.so) along with the native code.
- Breakpad Wrapper repo: https://code.rdkcentral.com/rdk/components/generic/breakpad_wrapper

Breakpad exception handler writes minidump to disk at exception time.

```c
void breakpad_ExceptionHandler()
{
    printf("\t\t\t ********ENTER breakpad_ExceptionHandler************** \n");

    static google_breakpad::ExceptionHandler* excHandler = NULL;

    excHandler = new google_breakpad::ExceptionHandler(google_breakpad::MinidumpDescriptor("/opt/minidumps"), NULL, breakpadDumpCallback, NULL, true, -1);

    printf("\t\t\t ******** breakpad_ExceptionHandler EXIT************** \n");
}
```
Breakpad-wrapper component compiles to create libbreakpadwrapper.so which needs to be linked with the RDK component for Breakpad integration.

Google Breakpad Integration With a RDK-B Component

**Step 1:** Create a .bbappend file with the following contents:

```
# generating minidumps symbols
inherit comcast-breakpad
DEPENDS += "breakpad breakpad-wrapper"
BREAKPAD_BIN_append = " CcspCMAgentSsp"

LDFLAGS += "-lbreakpadwrapper -lpthread -lstdc++"

# generating minidumps
PACKAGECONFIG_append = " breakpad"
```

**Step 2:** In the ssp_main.c file of component, add:

```
#ifdef INCLUDE_BREAKPAD
#include "breakpad_wrapper.h"
#endif
```

**Step 3:** In the main() of ssp_main.c, add:

```
breakpad_ExceptionHandler();
```

Other Debugging Techniques

1. CPE Logs - All logs are under /rdkblogs/logs directory
2. tcpdump - tcpdump -i erouter0 -s 65535 -w /tmp/data/cap.pcap
3. dmcli command
   - dmcli eRT setv Device.WiFi.SSID.1.SSID string test5
   - dmcli eRT getv Device.X_CISCO_COM_CableModem.MACAddr
   - dmcli eRT setv Device.ManagementServer.URL string http://tel_acs
4. Using gdb to debug core files
   - /usr/bin/gdb /usr/ccsp/lm/CcspLMLite <<corefile>> then pass bt

Logging

Presentation of system events, errors and application traces altogether in form of console messages or persistent log files constitutes the logging process.

Generating and reading system logs is an important aspect of system administration. The information in system logs can be used to detect hardware and software issues as well as application and system configuration errors. This information also plays an important role in assessing device health, security loopholes and incident response.

There are multiple levels of logging supported by RDK right from having the native Linux syslog (or more recently : journal logs) facility to having a dedicated logging facility such as RDK logger library.

- RDK-B Logger

Useful Links:

- RDK Utility - Breakpad