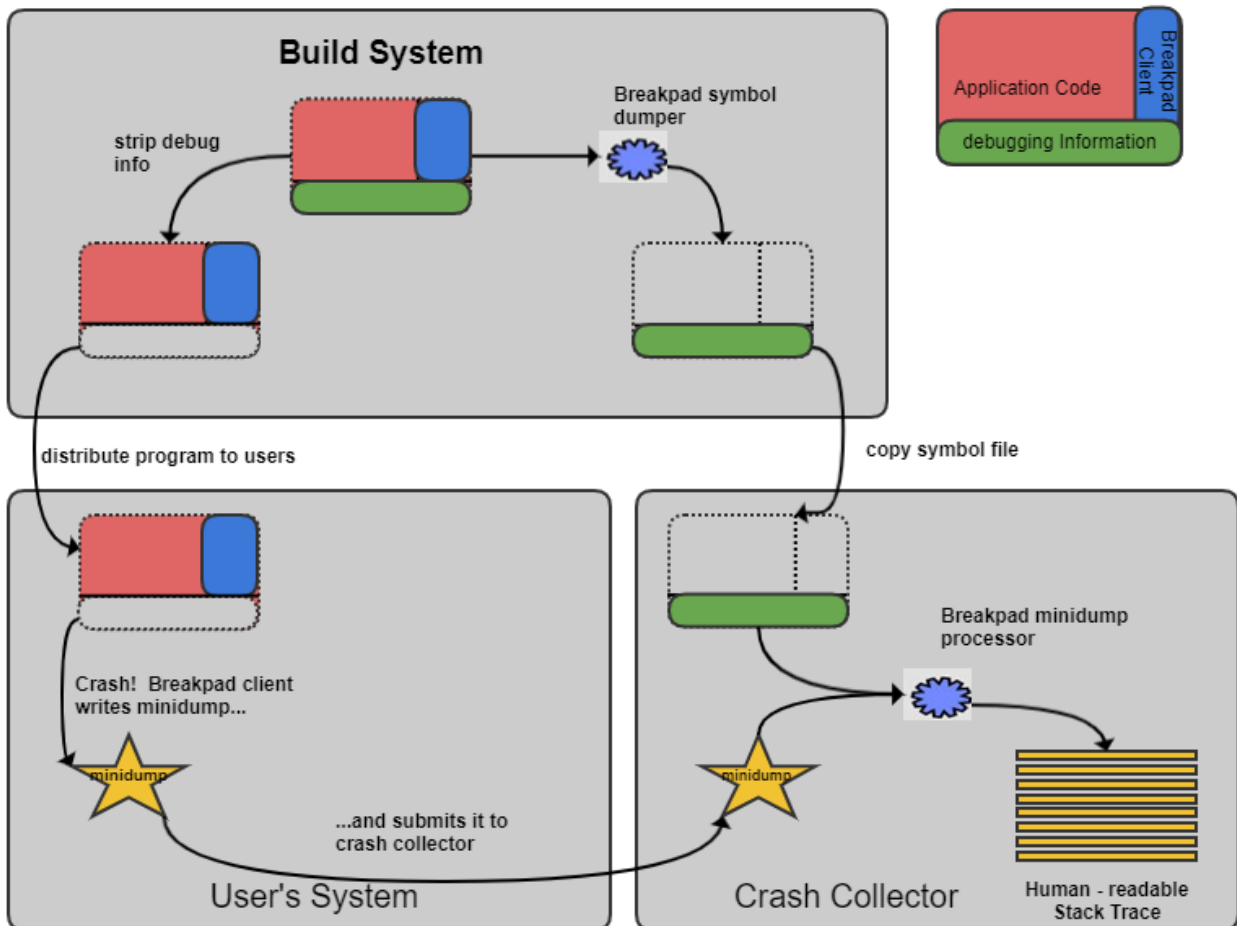


Google Breakpad

Breakpad is a library and tool suite that allows to distribute an application to users with compiler-provided debugging information removed. Breakpad library is linked with application which is executed on platform. When application crashes, it produces compact "minidump" files. These minidumps are send back to server and produce C and C++ stack traces.

Breakpad Capabilities

- Removes debug symbols from client code.
- (Client) Writes minidump file with thread context.
- (Client) Submits minidump to Crash Collector.
- (Crash Collector) Reconstructs human readable stack trace.



When compiled, the stack produces library (libbreakpad_client.a) which should be linked with the test application. When application crashes, it creates minidump files. There are breakpad utilities like dump_syms and minidump_stackwalker which are used to analyze the minidump.

Setting up Breakpad

Cross-Compile Breakpad library for target board (eg:RPI)

- git clone <https://chromium.googlesource.com/breakpad/breakpad>

- ./configure --host=arm-rdk-linux-gnueabi --prefix=/usr
- make

After successful compilation, "libbreakpad_client.a" will be created in "src/client/linux/" directory.

Compile Breakpad for PC to get utilities to analyze minidump

- git clone <https://chromium.googlesource.com/breakpad/breakpad>
- ./configure
- make

After successful compilation, executables dump_syms and minidump_stackwalk will be created in "src/tools/linux/" directory.

Steps to link google breakpad and create minidump

1. Create a sample app
vim breakpad_exercise.c
2. Include header file for exception handler
#include "client/linux/handler/exception_handler.h"
3. Add breakpad handler in your application. This is the callback function which will be executed when a crash occurs:

```
static bool breakpadDumpCallback(const google_breakpad::MinidumpDescriptor&descriptor, void*
context, bool succeeded) {
    printf(" Crash occurred, Callback function called.\n ");
    return succeeded;
}
```

4. In main() function, add the handler and register it. Instantiate exception handler for breakpad with the path where minidumps will be created. Here, current directory (". / ") where sample app is present & executed is given as path.

```
static google_breakpad::ExceptionHandler* excHandler = NULL;
excHandler = new google_breakpad::ExceptionHandler(google_breakpad::MinidumpDescriptor("./"),
NULL, breakpadDumpCallback, NULL, true, -1);
```

5. Create a crash in a function and call this function in main()

```
void err_func(){
    char *p = NULL;
    *p = 'A';
    printf("Value of pointer p: %c\n", *p);
}
```

6. Link Breakpad library and include path in Makefile

```
PKG_CONFIG_PATH=../
all:breakpad_exercise.c
    @ $(CXX) -std=c++11 breakpad_exercise.c -g -o breakpad_exercise `pkg-config --cflags breakpad` -
L./client/linux/-lbreakpad_client-ld./-lpthread
```

7. Run the application (which crashes and minidump gets generated)

```
root@raspberrypi-rdk-hybrid:~# ls
breakpad_exercise
root@raspberrypi-rdk-hybrid:~# ./breakpad_exercise
Crash occurred, Callback function called.
Segmentation fault (core dumped)
root@raspberrypi-rdk-hybrid:~#
```

A minidump file will be generated in the same directory:

```
root@raspberrypi-rdk-hybrid:~# ls
40e9abf8-19cc-4b55-cd2bb29f-dbd37900.dmp  breakpad_exercise
root@raspberrypi-rdk-hybrid:~#
```

Analyze minidump

dump_syms

Breakpad tool “dump_syms” run on binaries to produce the text-format symbols. The minidump should be copied to server pc where dump_syms is present.

```
breakpad/src/tools/linux/dump_syms/dump_syms breakpad_exercise > breakpad_exercise.sym
```

Run below command on symbol file to get the first line:

```
head -n1 breakpad_exercise.sym
```

Output (for example):

```
MODULE Linux arm 73DC1FFAD46D0ECDC4988DBBD008BBC70 breakpad_exercise
```

In the ideal scenario, this symbol file will be extracted initially and uploaded to some server. The application/library without symbol will be deployed. Once crashed, the minidump will be generated which will be analyzed along with this symbol file to generate stack trace.

minidump_stackwalk

This utility will give meaningful trace from minidump and symbol file

Create directory of name of this string (code), as shown below:

```
mkdir -p symbols/breakpad_exercise/73DC1FFAD46D0ECDC4988DBBD008BBC70
```

Copy “breakpad_exercise.sym” file to the above path.

```
cp breakpad_exercise.sym symbols/breakpad_exercise/73DC1FFAD46D0ECDC4988DBBD008BBC70
```

Run minidump_stackwalk tool on minidump file as below to produce a symbolized stack trace

```
breakpad/src/processor/minidump_stackwalk 40e9abf8-19cc-4b55-cd2bb29f-dbd37900.dmp symbols/ > tracefile
```

```
Operating system: Linux
                  0.0.0 Linux 4.1.21 #1 SMP Wed May 17 06:33:42 UTC 2017 armv7l
CPU: arm
    ARMv1 ARM part(0x4100d030) features: half,thumb,fastmult,vfpv2,edsp,neon,
    vfpv3,tls,vfpv4,idiva,idivt,4 CPUs

GPU: UNKNOWN

Crash reason:  SIGSEGV
Crash address: 0x0
Process uptime: not available

Thread 0 (crashed)
 0 breakpad_exercise!err_func [breakpad_exercise.c : 13 + 0x8]
   r0 = 0x7e9a0bf0    r1 = 0x00000001    r2 = 0x00000041    r3 = 0x00000000
   r4 = 0x01a5cf10    r5 = 0x7e9a0b80    r6 = 0x0000987c    r7 = 0x00000000
   r8 = 0x00000000    r9 = 0x00000000    r10 = 0x76f5e000   r12 = 0x76e9463c
   fp = 0x7e9a0b6c    sp = 0x7e9a0b60    lr = 0x00009afc    pc = 0x00009a18
Found by: given as instruction pointer in context
 1 breakpad_exercise!main [breakpad_exercise.c : 23 + 0x2]
   r4 = 0x01a5cf10    r5 = 0x7e9a0b80    r6 = 0x0000987c    r7 = 0x00000000
   r8 = 0x00000000    r9 = 0x00000000    r10 = 0x76f5e000   fp = 0x7e9a0c04
   sp = 0x7e9a0b70    pc = 0x00009afc
Found by: call frame info
 2 libc-2.23.so + 0x16eb6
   r4 = 0x0001924c    r5 = 0x00000000    r6 = 0x0000987c    r7 = 0x00000000
   r8 = 0x00000000    r9 = 0x00000000    r10 = 0x76f5e000   fp = 0x7e9a0d54
   sp = 0x7e9a0c08    pc = 0x76c5ceb8
Found by: call frame info
```