Qualcomm Technologies, Inc.

Qualcomm Linux Debug Guide

80-70018-12 AC

May 2, 2025

Contents

1 Debug overview

A subsystem processes independently within its own execution environment on the Qualcomm® SoC. This section describes the tools, sample logs, and troubleshooting methods for impacted subsystems. Understanding the discrepancies in subsystems and methods for examining errors is useful for diagnosing and troubleshooting such anomalies.

The following table lists the important subsystems on the SoC.

Table: Subsystems on SoC

Subsystem	Description
Application processor subsystem (APSS)	This primary subsystem executes the Qualcomm Linux kernel as the high-level OS (HLOS).
Application digital signal processor (aDSP)	This subsystem, known as the low-power audio subsystem (LPASS), handles digital signal processing tasks, such as audio encoding/decoding, and voice recognition.
Compute digital signal processor (cDSP)	This subsystem, known as the compute DSP, performs compute-intensive tasks, such as neural network-related calculations.
Qualcomm® Trusted Execution Environment (TEE)/TrustZone (TZ)	This subsystem performs secure operations and leverages the Arm® TrustZone® architecture. For more information, see Qualcomm Linux Security Guide.
Wireless local area network processor subsystem (WPSS)	This subsystem connects to a Wi-Fi network, transmits and receives data packets, handles security protocols, and ensures stable network performance. For more information, see Qualcomm Linux Wi-Fi Guide.
Always On Processor (AOP)	This subsystem regulates the power on the device.

Software for all subsystems, except the application processor, is known as non-HLOS.

To understand the boot flow of the subsystems, see Qualcomm Linux Boot Guide.

Note: See Hardware SoCs that Qualcomm® Linux® supports.

1.1 Debug workflow

Identify the subsystem where the issue occurred first, as many subsystems are present on the Qualcomm SoC. This helps you debug the relevant subsystem.

The application processor, as the primary subsystem detects when other subsystems crash. For example, if the aDSP processor subsystem crashes, the kernel log captures the subsystem restart (SSR) crash error log. Therefore, to identify the subsystem that need debugging, first verify the kernel debug messages.

The following figure shows the workflow to identify errors and the components you need to debug.

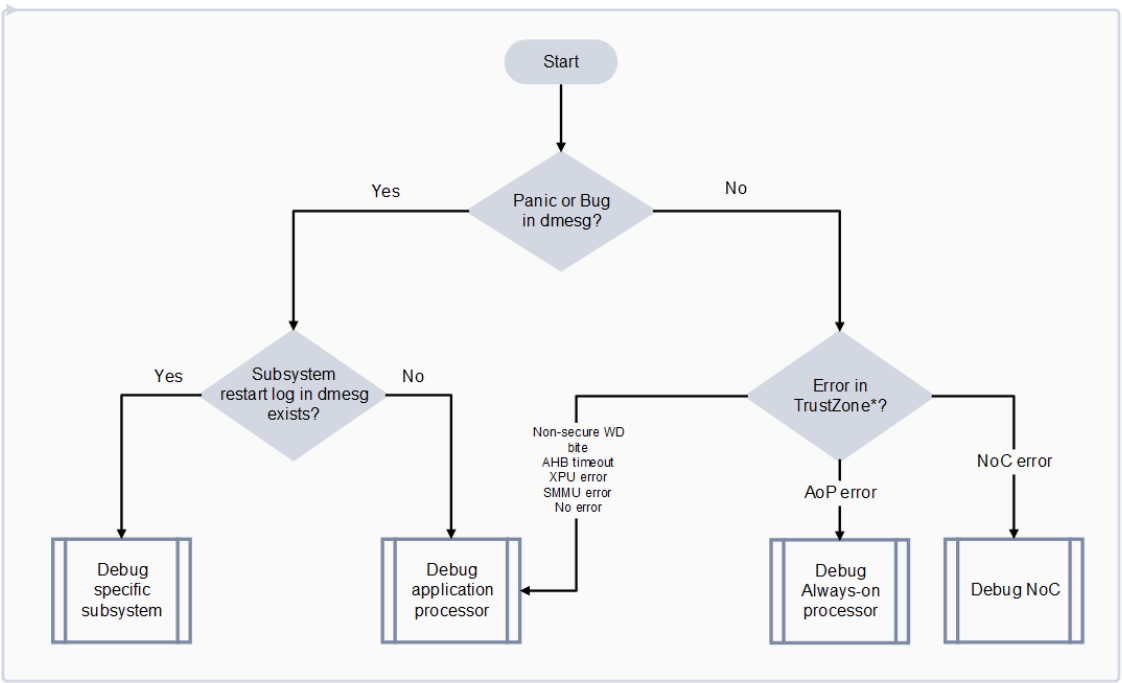


Figure : Workflow to identify the impacted subsystem

When the kernel dmesg logs show no panic signatures, verify the Qualcomm TEE diagnostic (diag) logs for errors such as nonsecure watchdog bite and network-on-chip (NoC) errors. The following sections describe example kernel messages that indicate kernel panic, bugs, and subsystem crash issues.

Identify kernel panic and bugs

You can determine from the kernel dmesg logs whether the reset was a kernel panic or a bug. The following are some example logs that indicate a kernel panic or a bug:

- Panic (Pattern: General error)

```
18.800936: <6> Kernel panic - not syncing: Fatal exception
18.800938: <6> SMP: stopping secondary CPUs
18.800947: <6> CPU0: stopping
Kernel panic - not syncing: Apps watchdog Bark received!
```

- Bug (Pattern 1: Forced crash due to memory corruption)

```
12.899532: <6> BUG kmalloc-128 (Not tainted): Redzone overwritten
12.905418: <6> ---------------------------------------------------------------------------
```

- Bug (Pattern 2: Forced crash from driver)

```
320.510769: <6> ------------[ cut here ]------------
320.510781: <6> kernel BUG at /local/mnt/workspace/lnxbuild/project/trees_in_
use/free_tree_platform_manifest_refs_tags/drivers/platform/msm/ipa/ipa_v3/ipa_qmi_
service.c:955!
```

- Bug (Pattern 3: Forced crash due to lockup issue)

```
[ 180.993861] BUG: spinlock lockup suspected on CPU#0, swapper/0/0
[ 180.993883] lock: stop_lock+0x0/0x18, .magic: dead4ead, .owner: swapper/6/0,
.owner_cpu: 6
[ 181.015629] Causing a watchdog bite!
```

Note: Most bugs typically result in a kernel panic. For more information about debugging kernel issues, see Qualcomm Linux Kernel Guide.

Identify subsystem crash issues

You can verify the kernel dmesg logs to determine whether the restart was due to a subsystem crash. For example, the following log indicates that the aDSP subsystem crashed, according to the `qcm6490.dtsi` (node: remoteproc_adsp: remoteproc@3000000).

```
0x000000000A27652C | 5198.790423: qcom_q6v5_pas 3000000.remoteproc: fatal error received: err_inject_crash.c:413:Crash injected via Diag
0x000000000A276689 | 5198.801061: remoteproc remoteproc2: crash detected in 3000000.remoteproc: type fatal error
0x000000000A2767A1 | 5198.809602: remoteproc remoteproc2: handling crash #1 in 3000000.remoteproc
0x000000000A27688E | 5198.816837: remoteproc remoteproc2: recovering 3000000.remoteproc
0x000000000A276971 | 5198.823784: qcom_q6v5_pas 8a00000.remoteproc: subsystem event rejected
```

Identify system issues from Qualcomm TEE logs

This feature is available to licensed developers with authorized access. For sample logs that indicate errors in the Qualcomm TEE, see Qualcomm Linux Debug Guide - Addendum.

If the Qualcomm TEE diag logs shows no error, the issue may be due to a secure watchdog bite issue. For more information about watchdog issues, see Hardware reset.

For more information about the debugging issues in Qualcomm TEE, see Qualcomm Linux Security Guide.

1.2 Debug methods

The following are the two debugging approaches:

- **On-target debugging**

  On-target debugging resolves software issues. This approach allows access to most information directly from the live device. Connect the Linux host to the live device using SSH. To set up SSH, see Sign in using SSH.

  For more information about various on-target debug features, see On-target debug features.

- **Off-target debugging**

  This approach uses logs instead of an actual device, making it an efficient method of debugging. You can debug using memory dumps or logging tools and various types of logs for offline debugging. As the RAM dump captures most of the memory regions, but only limited hardware register information, debugging a hardware-related issue can be a

challenge.

For more information about various off-target debug features, see Off-target debug features.

1.3 On-target debug features

You can use the following debug features on the device at runtime:

- Perf utility
- Subsystem restart
- Force subsystem reset
- Debug using OpenOCD

Perf utility

The Perf utility in Linux facilitates performance analysis and profiling. The Linux kernel includes this utility at the `tools/perf` directory.

You can use the Perf utility to debug various aspects of system behavior, including CPU performance counters, tracepoints, kprobes, and uprobes (for dynamic tracing). The following table lists the key features of the Perf utility.

Table : Key features of Perf utility

Feature	Description
CPU performance counters	These CPU hardware registers track events such as executed instructions, cache misses, and branch mispredictions. These events form the basis for profiling applications and identifying performance bottlenecks.
Tracepoints	You can place these tracepoints at logical locations in the code, such as system calls, network events, and file system operations. These tracepoints provide information such as timestamps and stack traces with minimal overhead.
Dynamic tracing	This feature dynamically creates tracepoints using the kprobes and uprobes frameworks, allowing tracing in both kernel space and user space.

For more information about the Perf utility, see the following resources:

- https://perf.wiki.kernel.org/index.php/Main_Page
- https://github.com/Linaro/OpenCSD/blob/master/HOWTO.md
- https://docs.kernel.org/trace/coresight/index.html

- https://docs.kernel.org/trace/coresight/coresight-perf.html

Subsystem restart

Subsystems can restart independently when they crash without requiring a device restart, called a subsystem restart (SSR). The SSR feature uses the remoteproc framework available in the Linux kernel.

To enable a full memory dump for debugging purposes, disable the SSR feature by default. It's recommended to use the SSR feature on commercial devices and not during the development phase. For more information about the SSR feature, see Remoteproc subsystem.

Enable the SSR feature to generate the SSR dump on a subsystem restart. Consider using the SSR dump for debugging, as its size is smaller than the full memory dump. However, you need the entire RAM dump to debug some subsystem crashes.

For more information about how to enable and capture subsystem dumps, see Subsystem dumps.

Force subsystem reset

Use this feature to debug use cases that require restarting a subsystem.

This feature is available to licensed developers with authorized access. To forcibly reset the subsystem, you can run the `diag` command using the QXDM Professional™. For more information about QXDM Professional commands and their usage, see Qualcomm Linux Debug Guide - Addendum.

1.4 Off-target debug features

You can use RAM dump or QXDM Professional for offline debugging.

RAM dump

The RAM dump is a snapshot of the entire memory at the time of failure. You can analyze the RAM dump using various tools, including Qualcomm® Crash Analysis Portal (QCAP), RAMParser, Lauterbach TRACE32 simulator, and crash utility.

For more information about how to collect and parse RAM dumps, see Collect and parse RAM dump.

QXDM Professional

Use QXDM Professional to debug various subsystems.

This tool is available to licensed developers with authorized access. For more information about QXDM Professional commands and their usage, see Qualcomm Linux Debug Guide - Addendum.

1.5 Common system issues

You can categorize system issues into one of the following types:

- Linux kernel space issues
- System issues
  - Watchdog issues
  - Bus hang and timeout issues
  - Hardware reset issues

1.6 Miscellaneous issues

You may run into the following miscellaneous issues:

- Device-freeze issues due to software bugs
- Linux application-related issues
- Random resets due to hardware issues related to the PCB

To debug Linux kernel-related issues, see Debug Linux kernel space issues.

To debug Linux application-related issues, see Debug Linux user space issues.

2 Debug Linux user space issues

The following figure shows the open-source tools, dumps, and logs you can use to debug issues related to Linux user space applications.

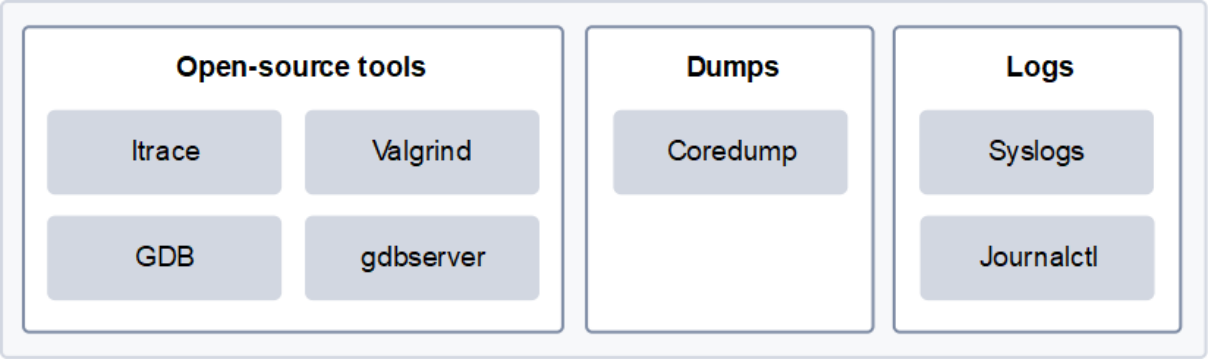


Figure : Resources to debug issues in the Linux user space

Enable the supported open-source debug tools using the `packagegroup-core-tools-debug.bbappend` BitBake file. You can find this BitBake file at the `layers/meta-qcom-hwe/recipes-devtools/packagegroups/` directory. The following sections describe the procedure to enable and use the open-source tools such as ltrace.

2.1 Debug using ltrace

By default, the build incorporates ltrace.

Prerequisite

Set up SSH. For instructions see Sign in using SSH.

Procedure

To debug with ltrace, run the following commands using SSH:

1. Run the process status command:

   ```
   ps -ef
   ```

   Sample output:

```
339 rpc 0:00 /usr/sbin/rpcbind -w -f
345 root 0:00 /lib/systemd/systemd-journald
373 root 0:00 [kworker/3:6-mm_]
```

2. Identify the process id (PID) of the process that you want to debug.
3. Run the `ltrace` command:

```
ltrace -p 345
```

In this example, `345` is the PID.

Sample output:

```
journal_file_close(0x25e30170, -1, 0, 0 <unfinished ...>
sockaddr_un_unlink(0x25e30170, -1, 0, 0
<unfinished ...>
sd_journal_close(0x25e31c00, 1, 3, 0x409e70) = 0
sd_journal_close(0x25e31d50, 1, 0, 1) = 0
<... sockaddr_un_unlink resumed> ) = 1
<... journal_file_close resumed> ) = 1
journal_file_close(0x25e30170, -1, 0, 0 <unfinished ...>
sockaddr_un_unlink(0x25e30170, -1, 0, 0 <unfinished ...>
sd_event_now(15, 0x41ed20, 10, 64) = 10
openat64(2, 0x41ed20, 10, 64 <unfinished ...>
clock_gettime(2, 0x41ed20, 10, 64) = 6
```

For more information, see Linux manual page.

2.2 Debug using Valgrind

The Valgrind tool helps detect memory-related errors that are common in C and C++ programs. These errors lead to crashes and unpredictable behavior, such as memory leaks and memory corruption. By default, the build incorporates the Valgrind tool.

Prerequisite

Set up SSH, for instructions see Sign in using SSH.

Procedure

To use the Valgrind tool, do the following:

1. Push debug symbols to the device. For instructions, see Configure debug symbols.

2. Run the following Valgrind tool command on the device:

```
valgrind --tool=memcheck --leak-check=yes --show-reachable=yes --num-callers=20 --track-fds=yes /usr/bin/tqftpserv
```

Sample output:

```
==622== Memcheck, a memory error detector
==622== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==622== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==622== Command: /usr/bin/tqftpserv
==622==

^C==622==
==622== Process terminating with default action of signal 2 (SIGINT)
==622== at 0x498EF38: select (select.c:69)
==622== by 0x4013CB: ??? (in /usr/bin/tqftpserv)
==622== by 0x48DB1AF: (below main) (libc_start_call_main.h:58)
==622==
==622== FILE DESCRIPTORS: 4 open (3 std) at exit.
==622== Open pf-42 socket 3:
==622== at 0x49984CC: socket (syscall-template.S:120)
==622== by 0x4891293: qrtr_open (in /usr/lib/libqrtr.so.1.0)
==622== by 0x40129B: ??? (in /usr/bin/tqftpserv)
==622== by 0x48DB1AF: (below main) (libc_start_call_main.h:58)
==622==
==622==
==622== HEAP SUMMARY:
==622== in use at exit: 0 bytes in 0 blocks
==622== total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==622==
==622== All heap blocks were freed -- no leaks are possible
==622==
==622== For lists of detected and suppressed errors, rerun with: -s
==622== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==622== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

For more information, see Valgrind.

2.3 Debug using GDB

By default, the build doesn't enable the GNU debugger (GDB). To enable GDB, do the following on the Linux host:

1. Go to the `layers/meta-qcom-hwe/recipes-devtools/packagegroups/` directory and open the `packagegroup-core-tools-debug.bbappend` file.
2. Change the `packagegroup-core-tools-debug.bbappend` file to add `gdb` in the package list. Skip this step if the package name is already in the package list.
3. Ensure that the build incorporates the debug symbols.

   ```
   readelf --debug-dump=decodedline <BIN_FILE>
   ```

   Or

   ```
   objdump --syms <BIN_NAME> | grep -i 'debug'
   ```

   If the debug symbols aren't enabled, compile all the required executables or shared libraries with the `-g` CFLAG. To push debug symbols to the device, see Configure debug symbols.
4. Recompile and flash the build on the device.

   To debug using GDB, run the following command on the device:

   ```
   gdb --pid 502
   ```

Sample output:

```
gdb: warning: Couldn't determine a path for the index cache directory.
GNU gdb (GDB) 11.2
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "aarch64-qcom-linux".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
Attaching to process 502
Reading symbols from /usr/bin/tqftpserv...
(No debugging symbols found in /usr/bin/tqftpserv)
Reading symbols from /usr/lib/libqrtr.so.1...
(No debugging symbols found in /usr/lib/libqrtr.so.1)
Reading symbols from /lib/libc.so.6...
Reading symbols from /lib/.debug/libc.so.6...
Reading symbols from /lib/ld-linux-aarch64.so.1...
Reading symbols from /lib/.debug/ld-linux-aarch64.so.1...
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/libthread_db.so.1".
0x0000ffffab00ef34 in __GI___select (nfds=4, readfds=0xffffc2ec14f8,
writefds=0x0, exceptfds=0x0, timeout=0x0) at
../sysdeps/unix/sysv/linux/select.c:69
69 ../sysdeps/unix/sysv/linux/select.c: No such file or directory.
(gdb)
(gdb)
(gdb)
(gdb)
```

The following table lists the commonly used GDB commands.

Table : Common GDB commands

Command	Description
`(gdb) bt`	Provides a backtrace of the current thread.
`(gdb) info threads`	Lists the IDs of currently known threads.
`(gdb) thread 2`	Switches to thread 2.
`(gdb) where`	Shows the current line number and the function that you are in.
`(gdb) thread apply all bt full`	Provides a backtrace of all threads.
`(gdb) info sharedlibrary`	Lists the names of shared libraries used.
`(gdb) info reg`	Lists the CPU registers.

For more information, see Linux manual page.

2.4 Collect coredump

A coredump is a memory snapshot of a user-space process that allows you to analyze the cause of a process crash.

The system collects coredumps according to the Yocto Linux standard for all user-space process crashes. By default, the build enables coredumps. The device saves the generated coredump at the `/var/coredump` directory.

To verify the location of the coredump, run the following command:

```
cat /proc/sys/kernel/core_pattern
```

Sample output:

```
/var/coredump/%e.core
```

The size of the coredump must be greater than 0 (zero). To verify the size of the coredump, run the following command:

```
ulimit -c
```

Sample output:

```
unlimited
```

If the coredump isn't enabled, use the following file to enable it:

`layers/meta-qcom-distro/recipes-products/packagegroups/packagegroup-qcom.bb`.

```
RDEPENDS:packagegroup-support-utils = "\
   chrony \
   libinput \
   libinput-bin \
   libnl \
   libxml2 \
+++ procps \
   "
```

Note: Use this patch to enable coredumps, then rebuild and reflash the device.

Analyzing coredump using GDB

You can generate a coredump by forcibly killing a process. To do this, run the following commands:

```
ps -ef | grep -i 'tqftp*'
```

Sample output:

```
root 1024 1 0 17:46 ? 00:00:00 /usr/bin/tqftpserv
root 1047 934 0 17:47 pts/0 00:00:00 grep -i tqftp*
```

```
kill -11 1024
```

```
cd /var/coredump
```

```
ls
```

Sample output:

```
tqftpserv.core
```

The `tqftpserv.core` is the coredump file.

If the device already has the debug symbols, run the following commands to use the GDB tool on the coredump:

```
cd /usr/bin/
```

```
gdb tqftpserv /var/coredump/tqftpserv.core
```

Sample output:

```
gdb: warning: Couldn't determine a path for the index cache directory.
GNU gdb (GDB) 11.2
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
Type "apropos word" to search for commands related to "word"...
Reading symbols from tqftpserv...

warning: exec file is newer than core file.
[New LWP 1024]
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/libthread_db.so.1".
Core was generated by `/usr/bin/tqftpserv'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0 0x0000ffff8629ef34 in __GI___select (nfds=4,
    readfds=readfds@entry=0xffffd04ccb28, writefds=writefds@entry=0x0,
--Type <RET> for more, q to quit, c to continue without paging--
    exceptfds=exceptfds@entry=0x0, timeout=timeout@entry=0x0)
    at ../sysdeps/unix/sysv/linux/select.c:69
69 ../sysdeps/unix/sysv/linux/select.c: No such file or directory.
(gdb) bt
#0 0x0000ffff8629ef34 in __GI___select (nfds=4,
    readfds=readfds@entry=0xffffd04ccb28, writefds=writefds@entry=0x0,
    exceptfds=exceptfds@entry=0x0, timeout=timeout@entry=0x0)
    at ../sysdeps/unix/sysv/linux/select.c:69
#1 0x00000000004013cc in main (argc=<optimized out>, argv=<optimized out>)
    at tqftpserv.c:552
Backtrace stopped: previous frame identical to this frame (corrupt stack?)
(gdb)
```

For more information, see Linux manual page.

2.5 Debug using gdbserver

The gdbserver utility allows you to run the GDB tool remotely from the Linux host. This utility is helpful in debugging issues when there is a storage constraint for debug symbols on the device.

Enable gdbserver on device

By default, the build doesn't enable gdbserver. To enable the gdbserver, do the following:

1. Go to the `layers/meta-qcom-hwe/recipes-devtools/packagegroups/` directory and open the `packagegroup-core-tools-debug.bbappend` file.
2. Change the `packagegroup-core-tools-debug.bbappend` file to add the `gdbserver` in the package list.
3. Recompile and flash the build on the device.

Configure gdbserver on device

To configure the gdbserver on the device, run the following commands using SSH:

```
mount -o rw,remount /
```

```
gdbserver :8888 <path-to-binary>
```

For example:

```
gdbserver :8888 /usr/bin/tqftpserv
```

Sample output:

```
sh-5.1# gdbserver :8888 /usr/bin/tqftpserv
Process /usr/bin/tqftpserv created; pid = 3214
Listening on port 8888
Remote debugging from host ::ffff:127.0.0.1, port 52814
Remote side has terminated connection.  GDBserver will reopen the connection.
Listening on port 8888
Remote debugging from host ::ffff:127.0.0.1, port 47252
```

The sample output indicates that the device is ready to communicate with the Linux host.

Configure gdb on a Linux host

To configure the gdb tool on a Linux host, do the following:

1. Install gdb and gdb-multiarch tools.

```
sudo apt-get install gdb gdb-multiarch
```

2. Create a debug directory to capture all the symbols from the build.

```
mkdir test_gdbserver
```

3. Copy the rootfs having the debug symbols from the build location to the `test_gdbserver` debug directory.

```
cp -f <ENTER_PATH>/build-qcom-wayland/tmp-glibc/deploy/images/
<chipset>/qcom-multimedia-image-<chipset>-dbg.rootfs.tar.bz2
test_gdbserver
```

```
cd test_gdbserver
```

```
tar -xvf qcom-multimedia-image-<chipset>-dbg.rootfs.tar.bz2
```

Note: While running these commands, replace `<chipset>` with the appropriate value as specified in the following table.

Chipset	Value
QCS6490	
QCS5430	`qcm6490`
QCS8275	`qcs8300`
QCS9075	`qcs9100`
QCS615	`qcs615`

The `test_gdbserver` is now ready with the symbols.

4. Start the gdb-multiarch tool using the `gdb-multiarch <EXECUTABLE_PATH>` command.

   For example,

```
gdb-multiarch test_gdbserver/usr/bin/tqftpserv
```

   The gdb-multiarch tool starts on the gdb console.

5. Run the following commands on the gdb console:

```
set gnutarget elf64-littleaarch64
```

Set `sysroot <path-to-new-debug-directory>`.

```
set sysroot <ENTER_PATH>/test_gdbserver
```

```
target remote <target_ip>:8888
```

```
bt
```

```
b
```

```
info threads
```

The following screenshots show the sample output of the configuration process.

```
sh-5.1# ps -ef
PID   USER     TIME  COMMAND
    1 root      0:04 {systemd} /sbin/init
    2 root      0:00 [kthreadd]
    3 root      0:00 [pool_workqueue_]
    4 root      0:00 [kworker/R-rcu_g]
    5 root      0:00 [kworker/R-rcu_p]
```

```
$ gdb-multiarch /test_gdbserver/usr/bin/.debug/tqftpserv
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.2) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later http://gnu.org/licenses/gpl.html
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
http://www.gnu.org/software/gdb/bugs/.
Find the GDB manual and other documentation resources online at:
        http://www.gnu.org/software/gdb/documentation/.
```

```
Reading symbols from /test_gdbserver/usr/bin/.debug/tqftpserv...
(gdb) set gnutarget elf64-littleaarch64
(gdb) set sysroot /test_gdbserver
(gdb) target remote 10.92.168.78:8888
Remote debugging using 10.92.168.78:8888
warning: while parsing target description (at line 71): Vector "v8bf16" references undefined
type "bfloat16"
warning: Could not load XML target description; ignoring
warning: Unable to find dynamic linker breakpoint function.
Reading symbols from /test_gdbserver/lib/ld-linux-aarch64.so.1...
(no debugging symbols found)...done.
Reading symbols from /test_gdbserver/lib/.debug/ld-linux-aarch64.so.1...
Loaded symbols for /test_gdbserver/lib/.debug/ld-linux-aarch64.so.1
warning: no loadable sections found in added symbol-file system-supplied DSO at
0xfffff7ffb000
0x0000fffff7fda870 in _start ()
from /test_gdbserver/lib/.debug/ld-linux-aarch64.so.1
(gdb) bt
#0 0x0000fffff7fda870 in _start ()
from /test_gdbserver/lib/.debug/ld-linux-aarch64.so.1
#1 0x0000000000000000 in ?? ()
(gdb) b
Breakpoint 1 at 0xfffff7fda870
(gdb) info threads
Id Target Id Frame
* 1 Thread 3212 0x0000fffff7fda870 in _start ()
(gdb)
```

For more information, see Linux manual page.

2.6 Debug with user space logs

Use the following logs to debug issues in the user space through SSH on the device.

Syslogs

To verify if the device generates syslogs, run the following command:

```
cat /var/log/user.log
```

If the device doesn't generate syslogs, run the following command:

```
tail -f /var/log/messages
```

Sample log:

```
Apr 28 17:42:30 qcm6490 daemon.info avahi-daemon[516]: No service file found in /etc/avahi/services.
Apr 28 17:42:30 qcm6490 daemon.info avahi-daemon[516]: Joining mDNS multicast group on interface lo.IPv6 with address ::1.
Apr 28 17:42:30 qcm6490 daemon.info avahi-daemon[516]: New relevant interface lo.IPv6 for mDNS.
Apr 28 17:42:30 qcm6490 daemon.info avahi-daemon[516]: Joining mDNS multicast group on interface lo.IPv4 with address 127.0.0.1.
Apr 28 17:42:30 qcm6490 daemon.info avahi-daemon[516]: New relevant interface lo.IPv4 for mDNS.
```

journalctl logs

To generate the systemd journalctl logs, run the following command:

```
journalctl -ef
```

Sample log:

```
Apr 28 17:42:28 qcm6490 kernel: spmi-temp-alarm
c440000.spmi:pmic@2:temp-alarm@2400: error -ENXIO: IRQ index 0 not found
Apr 28 17:42:28 qcm6490 kernel: qcom-spmi-adc5 c440000.spmi:pmic@2:adc@3100:
Invalid dig version read -19
Apr 28 17:42:28 qcm6490 kernel: qcom-spmi-adc5 c440000.spmi:pmic@2:adc@3100: error
-ENODEV: adc get dt data failed
Apr 28 17:42:28 qcm6490 kernel: qcom-spmi-adc5 c440000.spmi:pmic@0:adc@3100: error
-EINVAL: adc get dt data failed
Apr 28 17:42:28 qcm6490 kernel: qcom-spmi-adc5: probe of
c440000.spmi:pmic@0:adc@3100 failed with error -22
Apr 28 17:42:28 qcm6490 kernel: dwc3 a600000.usb: Adding to iommu group
Apr 28 17:42:29 qcm6490 systemd[1]: First Boot Complete was skipped because of a failed
condition check (ConditionFirstBoot=yes).
Apr 28 17:42:29 qcm6490 systemd[1]: Reached target Hardware activated USB gadget.
```

2.7 Configure debug symbols

You need debug symbols to parse coredumps. By design, Yocto Linux compiles a package and splits it into several packages. For example, if the `hello_0.1.bb` file compiles the `hello.cpp` file, it generates several packages.

The following table lists the packages that are relevant to the coredump.

Table : Packages required for parsing coredump

Package	Description
`hello_0.1-r0_armv8-2a.ipk`	This package includes a stripped executable and is the only package included in the image for the device.

Package	Description
`hello-dbg_0.1-r0_armv8-2a.ipk`	This package includes the debug symbols and is never packed in the image. The debug package (-dbg) significantly increases the image size, causing problems when including it. Apart from debugging, this package has no runtime value. Therefore, as a strategy, Yocto doesn't include any -dbg package in the image.
`hello-dev_0.1-r0_armv8-2a.ipk`	This package includes the exported headers and libraries that dependent modules use during compilation.

According to Yocto Linux standards, the build location path: `tmp-glibc/deploy/ipk/armv8-2a` stores the debug symbols. For example, `tqftpserv-dbg_0.0+0+de42697a24-r0_armv8-2a.ipk` package includes the debug symbols for the `/usr/bin/tqftpserv` directory.

To push the debug symbols to the device, do the following using SSH:

1. Remount the rootfs.

```
mount -o rw,remount /
```

2. Using the `scp` command, push the debug symbols (`tqftpserv-dbg_0.0+0+de42697a24-r0_armv8-2a.ipk`) to the device at any available partition such as `/data/`.

```
chmod 777 /data/tqftpserv-dbg_0.0+0+de42697a24-r0_armv8-2a.ipk
```

```
cd data
```

3. Install debug symbols on the device.

```
opkg install --nodeps tqftpserv-dbg_0.0+0+de42697a24-r0_armv8-2a.ipk
```

After you push the debug symbols to the device, the system saves the symbols at the path of the executable directory in the `.debug` directory. For example, if the executable is in the `/usr/bin/tqftpserv` directory, then the system saves the debug symbols in the `/usr/bin/.debug` directory.

To identify the debug symbols that are available on the device, run the following commands:

```
cd /usr/bin/.debug
```

```
ls
```

Sample output:

```
gencat getent locale pcprofiledump sprof zdump
getconf iconv makedb pldd tqftpserv
```

2.8 Reboot commands

The user-space reboot framework triggers the reboot using the kernel driver (`drivers/firmware/psci/psci.c`). You can run the following reboot commands using SSH:

Table : Reboot commands

Command	Description
`reboot edl`	Use this command to restart the device to the Emergency download (EDL) mode. For more information about the EDL mode, see Qualcomm Linux Build Guide.
`reboot bootloader`	Use this command to restart the device to the Fastboot mode. For more information about the Fastboot mode, see Qualcomm Linux Boot Guide.

You can use the following optional parameters with the `reboot` command.

```
Reboot the system.



Options:
      --help      Show this help
      --halt      Halt the machine
  -p --poweroff   Switch off the machine
      --reboot    Reboot the machine
  -f --force      Force immediate halt/power-off/reboot
  -w --wtmp-only  Don't halt/power-off/reboot, just write wtmp record
  -d --no-wtmp    Don't write wtmp record
      --no-wall   Don't send wall message before halt/power-off/reboot
```

3 Debug Linux kernel space issues

At a high-level, you can categorize issues in the kernel space as kernel panic and bugs. The following figure shows the logs, command-line options, and dumps that you can use to debug issues in the kernel space.

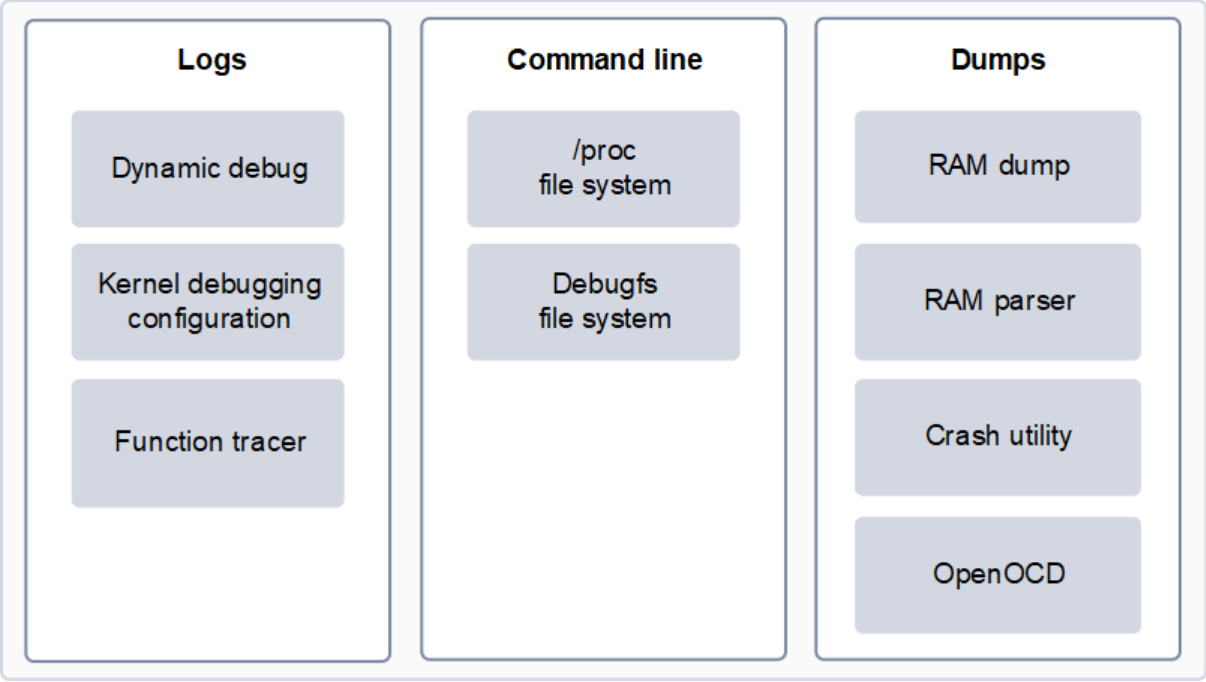


Figure : Resources to debug issues in kernel space

You can generate kernel logs using the `dmesg` command.

To debug issues in the kernel space, it's recommended to use the `debug` build. For more information about how to generate the `debug` build, see Qualcomm Linux metadata layers overview.

For more information about kernel source configuration files, see Qualcomm Linux Kernel Guide.

3.1 Enable dynamic debug

The debugfs file system allows you to debug the kernel by enabling logs at runtime. When verifying a particular scenario, use the debugfs file system to enable logs for the specific time.

By default, the system disables the dynamic debugfs. To enable debugfs in the kernel, do the following:

1. Enable the `CONFIG_DYNAMIC_DEBUG` kernel configuration option.
2. Recompile and reflash the build.
3. To mount the debugfs file system, run the following commands:

   ```
   mount -o rw,remount /
   ```

   ```
   mount -t debugfs none /sys/kernel/debug
   ```

   To verify if the dynamic debug is enabled ensure the following node exists:

   ```
   cd /sys/kernel/debug/dynamic_debug
   ```

   If the node exists, you must verify the defined logs:

   ```
   cat /sys/kernel/debug/dynamic_debug/control
   ```

4. Enable the debug log for the files or the function that requires debugging.

   For example:

   - To enable all debugfs logs in the `mdp.c` file, run the following command:

     ```
     echo 'file mdp.c +p' > /sys/kernel/debug/dynamic_debug/
     control
     ```

   - To enable the log at line 2921 in the `mdp.c` file, run the following command:

     ```
     echo 'file mdp.c line 2921 +p' > /sys/kernel/debug/dynamic_
     debug/control
     ```

5. Verify logs using the `dmesg` command or run the following command:

   ```
   cat /proc/kmsg
   ```

   For more information, see Documentation/dynamic-debug-howto.txt.

3.2 Enable kernel debugging options

You can enable the kernel configuration options to debug various issues such as memory leak, lock-related, and mutex problems. To see the available kernel debugging options, invoke `menuconfig`. For more information about kernel debugging options, see Kernel configurations. The following are some debugging options:

```
Kernel hacking
[*] Kernel debugging
[*] Detect Soft Lockups
[ ] Collect scheduler statistics
[*] Debug slab memory allocations
[*] Memory leak debugging
[*] Mutex debugging, deadlock detection
[*] Spinlock debugging
[*] Sleep-inside-spinlock checking
[ ] kobject debugging
[ ] Highmem debugging
[ ] Compile the kernel with debug info
```

Note: If you enable any of these debugging options, the kernel slows down marginally. Therefore, if you notice any decrease in performance, disable the kernel debug configuration options.

3.3 Kernel configuration options

The following table lists the common kernel configuration options that are useful for debugging.

Table : Common kernel debug configuration options

Kernel debug configuration option	Description
CONFIG_DEBUG_LIST	This option turns on checks for performing standard linked list manipulations with the `list.h` header file. If the pointers don't match, the system prints a warning, followed by a `BUG_ON` crash.
CONFIG_PAGE_POISONING	This option fills the pages with the poison pattern `(PAGE_POISON 0xaa)`, after calling `free_pages()`.

Kernel debug configuration option	Description
CONFIG_DEBUG_PAGEALLOC	This option verifies the patterns before calling `alloc_pages()`.
CONFIG_DEBUG_USER	This option prints a message when the system kills a user-space process due to a segmentation fault (segfault) or an invalid instruction, such as `user_debug=31` in the `arch/arm/Kconfig.debug` file. Add the Kernel boot parameter to the `BoardConfig.mk` file.
CONFIG_DEBUG_SPINLOCK	This option identifies missing spinlock initialization and common spinlock errors, such as: • Waiting for more than one second on a spinlock • Freeing an already freed lock • Reinitializing a lock that was already used
CONFIG_DEBUG_MUTEXES	This option detects Mutex semantic violations.
DEBUG_LOCK_ALLOC	This option detects wrong freeing of live locks.
CONFIG_SLUB CONFIG_SLUB_DEBUG	This option performs extra checks to detect the corruption of internal kernel memory allocation structures by adding poison for `use-after-free (0x6b)` and `buffer-overflow-padding (0xbb)`.
Kernel debug configuration options for extra debugging that can be verbose and can make the system slow.	
CONFIG_DEBUG_ATOMIC_SLEEP	This option causes routines that may sleep to become noisy when they're called within the atomic sections.
DEBUG_SPINLOCK_SLEEP	This option causes routines that may sleep to become noisy when they're called with a spinlock held.
CONFIG_DEBUG_VM, CONFIG_DEBUG_HIGHMEM	This option provides an extra debugging support for virtual memory management corruption issues.
CONFIG_DEBUG_OBJECTS	This option tracks the lifetime of various objects and validates the operations on those objects.

The following are example logs indicating common memory issues.

List corruptions

Enabling the `CONFIG_DEBUG_LIST` option helps you identify the following crash signatures that indicate list corruption issues.

Sample crash signature 1

```
<4> WARNING: at kernel/lib/list_debug.c:60 __list_del_entry+0xa0/0xd0()
<6> list_del corruption. prev->next should be c6fc374c, but was c18d804c
<6> Modules linked in: adsprpc
<6> [<c010cc94>] (unwind_backtrace+0x0/0x138) from [<c018ad54>] (warn_
slowpath_common+0x4c/0x64)
<6> [<c018ad54>] (warn_slowpath_common+0x4c/0x64) from [<c018ae00>]
(warn_slowpath_fmt+0x30/0x40)
<6> [<c018ae00>] (warn_slowpath_fmt+0x30/0x40) from [<c03b3128>] (__list_
del_entry+0xa0/0xd0)
<6> [<c03b3128>] (__list_del_entry+0xa0/0xd0) from [<c01bc968>] (account_
entity_dequeue+0x84/0x94)
<6> [<c01bc968>] (account_entity_dequeue+0x84/0x94) from [<c01bdaa4>]
(dequeue_task_fair+0x64/0x190)
```

Sample crash signature 2

```
<4>WARNING: at kernel/lib/list_debug.c:52 __list_del_entry+0x8c/0xac()
<4> list_del corruption, e240c500->prev is LIST_POISON2 (00200200)
<4> Modules linked in: wlan(PO) cfg80211 adsp_loader
<4> [<c001498c>] (unwind_backtrace+0x0/0x11c)
from [<c0068c90>] (warn_slowpath_common+0x4c/0x64)
<4> [<c0068c90>] (warn_slowpath_common+0x4c/0x64)
from [<c0068d28>] (warn_slowpath_fmt+0x2c/0x3c)
<4> [<c0068d28>] (warn_slowpath_fmt+0x2c/0x3c)
from [<c023ef94>] (__list_del_entry+0x8c/0xac)
<4> [<c023ef94>] (__list_del_entry+0x8c/0xac)
from [<c023efc0>] (list_del+0xc/0x24)
<4> [<c023efc0>] (list_del+0xc/0x24)
from [<c043c78c>] (binder_thread_read+0x488/0xb70)
<4> [<c043c78c>] (binder_thread_read+0x488/0xb70)
from [<c043d094>] (binder_ioctl+0x220/0x5b8)
```

Sample crash signature 3

```
<4> WARNING: at kernel/lib/list_debug.c:47 __list_del_entry+0x90/0xb0()
<4> list_del corruption, d6ed3720->next is LIST_POISON1 (00100100)
<4> Modules linked in:
<4> [] (unwind_backtrace+0x0/0x12c) from [] (warn_slowpath_
common+0x4c/0x64)
<4> [] (warn_slowpath_common+0x4c/0x64) from [] (warn_slowpath_
fmt+0x2c/0x3c)
<4> [] (warn_slowpath_fmt+0x2c/0x3c) from [] (__list_del_entry+0x90/0xb0)
<4> [] (__list_del_entry+0x90/0xb0) from [] (list_del+0xc/0x28)
<4> [] (list_del+0xc/0x28) from [] (bam_mux_write_done+0x34/0x11c)
```

Spinlock corruption issues

Enabling the `CONFIG_DEBUG_SPINLOCK` and `CONFIG_DEBUG_MUTEXES` kernel configuration options helps you identify crash signatures that indicate spinlock corruption issues.

Crash signature 1

```
<0>BUG: spinlock lockup on CPU#1, ndroid.launcher/1071
<0> lock: 0xd5e8f480, .magic: dead4ead, .owner: <none>/-1, .owner_cpu: -1
<4> [<c0734d84>] (spin_dump+0x74/0x84) from [<c028f8fc>]
(do_raw_spin_lock+0x144/0x188)
<4> [<c028f8fc>] (do_raw_spin_lock+0x144/0x188) from [<c0332594>] (kgsl_
mmu_pt_get_flags+0x18/0x44)
<4> [<c0332594>] (kgsl_mmu_pt_get_flags+0x18/0x44) from [<c033c6c8>]
(adreno_ringbuffer_submitcmd+0x168/0x228)
<4> [<c033c6c8>] (adreno_ringbuffer_submitcmd+0x168/0x228) from
[<c033d9a4>] (sendcmd+0x3c/0x254)
```

Crash signature 2

```
<0> BUG: spinlock lockup on CPU#2, kworker/2:0H/2910
<0> lock: kpss_clock_reg_lock+0x0/0x10, .magic: dead4ead, .owner:
kworker/3:0H/2904, .owner_cpu: 3
<6> kworker/2:0H (2910): undefined instruction: pc=c0963098
kernel BUG at kernel/lib/spinlock_debug.c:95!

<4> [<c0963098>] (spin_dump+0x7c/0x94) from [<c038e030>]
(do_raw_spin_lock+0xcc/0x164)
<4> [<c038e030>] (do_raw_spin_lock+0xcc/0x164) from [<c0972228>] (_raw_
spin_lock_irqsave+0x20/0x28)
<4> [<c0972228>] (_raw_spin_lock_irqsave+0x20/0x28) from [<c011d4ac>] (__
kpss_mux_set_sel+0x14/0x80)
<4> [<c011d4ac>] (__kpss_mux_set_sel+0x14/0x80) from [<c011d54c>] (kpss_
mux_set_sel+0x18/0x20)
```

Slub poisoning issues

Enabling the `CONFIG_SLUB` and `CONFIG_SLUB_DEBUG` kernel configuration options helps you identify crash signatures that indicate a slub poisoning issue.

```
<3>[ 3438.930472]
=====================================================================
<3>[ 3438.937628] BUG kmalloc-64 (Tainted: G W O): Poison overwritten
<3>[ 3438.944223] ---------------------------------------------------------------
<3>[ 3438.944228]
<3>[ 3438.953861] INFO: 0xce308408-0xce30840b. First byte 0x0 instead of 0x6b
<3>[ 3438.960470] INFO: Allocated in kgsl_ioctl_drawctxt_create+0x2c/0x2ac age=4426
cpu=0 pid=21702
<3>[ 3438.968970] __slab_alloc.isra.37.constprop.43+0x4d4/0x534
<3>[ 3438.974435] kmem_cache_alloc_trace+0x240/0x258
<3>[ 3438.978947] kgsl_ioctl_drawctxt_create+0x2c/0x2ac
<3>[ 3438.983721] kgsl_ioctl+0xfc/0x324
<3>[ 3438.987108] do_vfs_ioctl+0x80/0x54c
<3>[ 3438.990669] sys_ioctl+0x38/0x5c
<3>[ 3438.993880] ret_fast_syscall+0x0/0x30
<3>[ 3438.997615] INFO: Freed in kgsl_release+0xb8/0xc0 age=10 cpu=3 pid=21842
<3>[ 3439.004298] __slab_free+0x30/0x308
<3>[ 3439.007770] kgsl_release+0xb8/0xc0
<3>[ 3439.011242] fput+0xcc/0x23c
<3>[ 3439.014106] filp_close+0x68/0x80
<3>[ 3439.017407] put_files_struct+0xd8/0x110
<3>[ 3439.021312] do_exit+0x164/0x860
<3>[ 3439.024524] do_group_exit+0x3c/0xb0
<3>[ 3439.028084] get_signal_to_deliver+0x2c4/0x59c
<3>[ 3439.032510] do_signal+0x90/0x480
<3>[ 3439.035809] do_notify_resume+0x50/0x5c
<3>[ 3439.039629] work_pending+0x24/0x28
<3>[ 3439.043101] INFO: Slab 0xc0fbcd20 objects=16 used=16 fp=0x (null) flags=0x0080
<3>[ 3439.050401] INFO: Object 0xce308400 @offset=1024 fp=0xce308900
<3>[ 3439.050405]
<3>[ 3439.057694] Bytes b4 ce3083f0: 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a
ZZZZZZZZZZZZZZZZ
<3>[ 3439.066457] Object ce308400: 6b 6b 6b 6b 6b 6b 6b 6b 00 00 00 00 6b 6b 6b 6b
kkkkkkkk....kkkk
<3>[ 3439.075051] Object ce308410: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b
kkkkkkkkkkkkkkkk
<3>[ 3439.083639] Object ce308420: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b
kkkkkkkkkkkkkkkk
<3>[ 3439.092234] Object ce308430: 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b a5
kkkkkkkkkkkkkkk.
```

```
<3>[ 3439.100828] Redzone ce308440: bb bb bb bb ....
<3>[ 3439.108463] Padding ce3084e8: 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a
ZZZZZZZZZZZZZZZZ
<3>[ 3439.117145] Padding ce3084f8: 5a 5a 5a 5a 5a 5a 5a 5a ZZZZZZZZ
<4>[ 3439.125144] [<c0014f78>] (unwind_backtrace+0x0/0x138) from [<c0150250>]
(check_bytes_and_report+0xc0/0xe4)
<4>[ 3439.134771] [<c0150250>] (check_bytes_and_report+0xc0/0xe4) from [<c015042c>]
(check_object+0x1b8/0x214)
<4>[ 3439.144237] [<c015042c>] (check_object+0x1b8/0x214) from [<c071b0bc>] (alloc_
debug_processing+0x7c/0x150)
<4>[ 3439.153782] [<c071b0bc>] (alloc_debug_processing+0x7c/0x150) from [<c071bcc0>]
(__slab_alloc.isra.37.constprop.43+0x4d4/0x534)
<4>[ 3439.165153] [<c071bcc0>] (__slab_alloc.isra.37.constprop.43+0x4d4/0x534) from
[<c015221c>] (kmem_cache_alloc_trace+0x240/0x258)
<4>[ 3439.176611] [<c015221c>] (kmem_cache_alloc_trace+0x240/0x258) from
[<c014668c>] (__get_vm_area_node.isra.26+0x84/0x174)
<4>[ 3439.187376] [<c014668c>] (__get_vm_area_node.isra.26+0x84/0x174) from
[<c0147038>] (get_vm_area_caller+0x44/0x4c)
<4>[ 3439.197615] [<c0147038>] (get_vm_area_caller+0x44/0x4c) from [<c014739c>]
(vmap+0x50/0x90)
<4>[ 3439.205866] [<c014739c>] (vmap+0x50/0x90) from [<c032a694>] (_kgsl_
sharedmem_page_alloc+0x238/0x3d4)
<4>[ 3439.214979] [<c032a694>] (_kgsl_sharedmem_page_alloc+0x238/0x3d4) from
[<c0323e44>] (_gpumem_alloc+0xb0/0xfc)
<4>[ 3439.224874] [<c0323e44>] (_gpumem_alloc+0xb0/0xfc) from [<c0323ed0>] (kgsl_
ioctl_gpumem_alloc_id+0x40/0x1a8)
<4>[ 3439.234684] [<c0323ed0>] (kgsl_ioctl_gpumem_alloc_id+0x40/0x1a8) from
[<c0321874>] (kgsl_ioctl+0xfc/0x324)
<4>[ 3439.244319] [<c0321874>] (kgsl_ioctl+0xfc/0x324) from [<c0167fe8>] (do_vfs_
ioctl+0x80/0x54c)
<4>[ 3439.252738] [<c0167fe8>] (do_vfs_ioctl+0x80/0x54c) from [<c01684ec>] (sys_
ioctl+0x38/0x5c)
<4>[ 3439.260986] [<c01684ec>] (sys_ioctl+0x38/0x5c) from [<c000eb00>] (ret_fast_
syscall+0x0/0x30)
<3>[ 3439.269400] FIX kmalloc-64: Restoring 0xce308408-0xce30840b=0x6b
```

Page poisoning issues

Enabling the `CONFIG_DEBUG_PAGEALLOC` and `CONFIG_PAGE_POISONING` kernel configuration options helps you identify crash signature that indicate a page poisoning issue.

```
<1> Unable to handle kernel paging request at virtual address aaaaaaae
<1> pgd = e98b4000
<1> [aaaaaaae] *pgd=00000000
<0> Internal error: Oops: 5 [#1] PREEMPT SMP ARM
<4> Modules linked in: adsp_loader exfat_fs(P) exfat_core(P)
<4> CPU: 1 Tainted: P W (3.4.0-628250-eng #1)
<4> PC is at pid_nr_ns+0xc/0x3c
<4> LR is at do_task_stat+0x248/0x83c
<4> pc : [<c00abda8>] lr : [<c018c15c>] psr: a0000093
```

3.4 CPU parameters

Various CPU parameters such as core frequency, CPU governor, and cpuidle states help you better understand the system and tune it as needed.

For more information about CPU parameters, see kernel documentation.

3.5 Memory usage

Linux uses a virtual memory system. Therefore, the addresses that the user program accesses don't correspond to the physical addresses that the hardware uses directly. Virtual memory introduces a layer of indirection, allowing programs to assign extra memory beyond the physically available memory.

Memory management implementation covers the following areas:

- Management of physical pages in the memory
- Buddy system to assign memory in large chunks
- Slab, slub, and slob allocators to assign smaller chunks of memory
- vmalloc mechanism to assign noncontiguous blocks of memory
- Address space of the processes

/proc file system

The `/proc` file system provides the following files:

- /proc/meminfo
- /proc/vmstat
- /proc/iomem
- /proc/vmallocinfo

/proc/meminfo

This file provides information about distribution and usage of memory. To view the contents of this file, run the following command:

```
cat /proc/meminfo
```

Sample output:

```
MemTotal: 3813532 kB
MemFree: 624836 kB
MemAvailable: 2098008 kB
Buffers: 40416 kB
Cached: 1484320 kB
SwapCached: 0 kB
Active: 1334816 kB
.
.
.
.
NFS_Unstable: 0 kB
Bounce: 0 kB
WritebackTmp: 0 kB
CommitLimit: 2431048 kB
Committed_AS: 99995284 kB
VmallocTotal: 258867136 kB
VmallocUsed: 0 kB
VmallocChunk: 0 kB
CmaTotal: 163840 kB
CmaFree: 1368 kB
```

For more information about the parameters available in this file, see /proc filesystem.

/proc/vmstat

This file shows detailed virtual memory statistics from the kernel. Most of the statistics are available only if you enable the `CONFIG_VM_EVENT_COUNTERS` option in the `init/Kconfig` file.

To view the contents of this file, run the following command:

```
cat /proc/vmstat
```

Sample output:

```
nr_free_pages 156290
nr_alloc_batch 132
nr_inactive_anon 108
nr_active_anon 165006
nr_inactive_file 212275
nr_active_file 168709
nr_unevictable 64
nr_mlock 64
nr_anon_pages 164982
nr_mapped 90366
nr_file_pages 381184:
:
unevictable_pgs_mlocked 0
unevictable_pgs_munlocked 0
unevictable_pgs_cleared 0
unevictable_pgs_stranded 0
unevictable_pgs_mlockfreed 0
```

For more information about the parameters available in this file, see Linux manual page.

/proc/iomem

This file shows the memory map of the system for its various device drivers. To view the contents of this file, run the following command:

```
cat /proc/iomem
```

Sample output:

```
007781b8-007791b7 : vmpm
010aa000-010abfff : tsens_physical
010ac000-010ac003 : pshold-base
010ad000-010aefff : tsens_physical
01680000-0168ffff : /soc/arm,smmu-anoc1@1680000
016c0000-016fffff : /soc/arm,smmu-anoc2@16c0000
01d0101c-01d0101f : sp2soc_irq_status
01d01024-01d01027 : sp2soc_irq_clr
01d01028-01d0102b : sp2soc_irq_mask
.
.
.
0caa0000-0caa3fff : jpeg_hw
0caa4000-0caa47ff : fd_core
0caa5000-0caa53ff : fd_misc
0cd00000-0cd3ffff : /soc/arm,smmu-mmss@cd00000
17817000-17817fff : msm-watchdog
17900000-1790dfff : msm-gladiator-erp
80000000-857fffff : System RAM
80080000-817fffff : Kernel code
82330000-82945fff : Kernel data
88f00000-8aafffff : System RAM
95300000-17e3bffff : System RAM
```

/proc/vmallocinfo

This file shows detailed information about virtual address allocation through `vmalloc` or `ioremap`. To view the contents of this file, run the following command:

```
cat /proc/vmallocinfo
```

Sample output:

```
0xbf000000-0xbf002000 8192 module_alloc_update_bounds+0xc/0x5c pages=1 vmalloc
0xbf004000-0xbf008000 16384 module_alloc_update_bounds+0xc/0x5c pages=3 vmalloc
0xee800000-0xef800000 16777216 iotable_init+0x0/0xb0 phys=36800000 ioremap
0xf0000000-0xf0002000 8192 of_iomap+0x30/0x38 ioremap
0xf0002000-0xf0004000 8192 of_iomap+0x30/0x38 ioremap
0xf0004000-0xf000c000 32768 gen_pool_add_virt+0x48/0xb8 pages= 7 vmalloc
0xf000c000-0xf000e000 8192 msm_pm_setup_saved_state+0xcc/0x1bc ioremap
..............................
0xf0174000-0xf0176000 8192 msm_cpu_status_probe+0xd8/0x20c ioremap
0xf0f24000-0xf0f28000 16384 _kgsl_sharedmem_page_alloc+0xa0/0x41c pages=3 vmalloc
0xf0f39000-0xf0f3e000 20480 _kgsl_sharedmem_page_alloc+0xa0/0x41c pages=4 vmalloc
0xf0f61000-0xf0f66000 20480 _kgsl_sharedmem_page_alloc+0xa0/0x41c pages=4 vmalloc
...........
0xfa400000-0xfa600000 2097152 iotable_init+0x0/0xb0 phys=fa00000 ioremap
0xfa71e000-0xfa71f000 4096 iotable_init+0x0/0xb0 phys=f991e000 ioremap
0xfefd8000-0xff000000 163840 pcpu_get_vm_areas+0x0/0x56c vmalloc
```

memblock interface

The `memblock` interface on the debugfs file system provides details about the available and reserved memory regions in the system. This interface provides the following files:

- /sys/kernel/debug/memblock/memory
- /sys/kernel/debug/memblock/reserved

/sys/kernel/debug/memblock/memory

This file provides information about all the available memory regions (HLOS and non-HLOS) visible to the Linux kernel. To determine the overall memory accessible to the kernel, calculate the difference between the start and end addresses of each of the region and sum these values to get the total occupied RAM. The remaining memory, which is the difference between the RAM size of the device and the occupied RAM, is the non-HLOS memory or the memory occupied by other subsystems.

To view the contents of this file, run the following command:

```
cat /sys/kernel/debug/memblock/memory
```

Sample output:

```
0: 0x0000000080000000..0x00000000857fffff
1: 0x0000000088f00000..0x000000008aafffff
2: 0x0000000095300000..0x000000017e3bffff
```

/sys/kernel/debug/memblock/reserved

This file provides information about all the reserved memory regions in the system.

To view the contents of this file, run the following command:

```
cat /sys/kernel/debug/memblock/reserved
```

Sample output:

```
0: 0x0000000080080000..0x0000000082944fff
1: 0x0000000083200000..0x0000000083259bb4
2: 0x0000000083400000..0x00000000839506c9
3: 0x00000000f5800000..0x00000000ffbfffff
4: 0x00000000ffff7000..0x00000000ffffefff
5: 0x00000000ffffff40..0x00000000ffffff77
6: 0x00000000ffffff80..0x00000000ffffffb7
7: 0x00000000ffffffc0..0x00000000ffffffff7
8: 0x0000000179258000..0x000000017d9fffff
9: 0x000000017da17000..0x000000017da1ffff
10: 0x000000017da20e00..0x000000017da26fff
11: 0x000000017da27300..0x000000017da2735f
12: 0x000000017da27380..0x000000017da273df
13: 0x000000017da27400..0x000000017da2755f
14: 0x000000017da27580..0x000000017da27587
15: 0x000000017da275c0..0x000000017da275c7
16: 0x000000017da29600..0x000000017da29924
17: 0x000000017da29940..0x000000017da29c64
18: 0x000000017da29c80..0x000000017da29fa4
19: 0x000000017da29fac..0x000000017da2a3f8
20: 0x000000017da2a3fc..0x000000017da2a42e
21: 0x000000017da2a430..0x000000017da2a45e
22: 0x000000017da2a460..0x000000017e3bffff
```

Memory leak

Enable the following configuration options to debug kernel memory leak issues:

- `CONFIG_DEBUG_KMEMLEAK=y`
- `CONFIG_DEBUG_KMEMLEAK_MEM_POOL_SIZE= 4000`
- `CONFIG_DEBUG_KMEMLEAK_DEFAULT_OFF=y`

By default, a kernel thread scans the memory every 10 minutes and prints the number of new unreferenced objects found. For example,

```
unreferenced object 0xec26f000 (size 4096):
comm "Binder_2", pid 4592, jiffies 8848 (age 336.710s)
hex dump (first 32 bytes):
ec 4d f8 c0 02 00 00 00 00 00 00 00 00 00 00 00 .M..............
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
backtrace:
[<c0126f70>] kmem_cache_alloc_trace+0x17c/0x238
[<c03059c4>] ddl_client_transact+0xd0/0x158
[<c0314a3c>] ddl_open+0x4c/0x194
[<c0302288>] vcd_init_client_context+0x14/0x9c
[<c02ffd10>] vcd_open_in_ready+0x3c/0x94
[<c02fd31c>] vcd_open+0x214/0x274
[<c031b244>] vid_dec_open_client+0x1d0/0x288
[<c031b3ac>] vid_dec_open+0x30/0x7c
[<c012ff70>] chrdev_open+0x10c/0x134
[<c012aa7c>] __dentry_open.isra.12+0x190/0x29c
[<c0138a78>] do_last.isra.29+0x690/0x6c0
[<c0138c70>] path_openat+0xb8/0x35c
[<c0138ff4>] do_filp_open+0x2c/0x78
[<c012b7a0>] do_sys_open+0xd8/0x170
[<c000df20>] ret_fast_syscall+0x0/0x30
[<ffffffff>] 0xffffffff
```

Disable the `KMEMLEAK` option at boot time by passing `KMEMLEAK=off` on the kernel command line.

For more information about the `kmemleak.txt` file, see kernel documentation.

The following extra kernel configuration options are available to track the allocator of each page of memory:

- `CONFIG_PAGE_OWNER`

- `CONFIG_PAGE_OWNER_ENABLE_DEFAULT`
- `CONFIG_PAGE_EXTENSION`

Enabling these options to parse all pages can help identify several allocations, which may indicate a memory leak issue.

Identify memory corruption issues

Enable the following kernel configuration options to identify memory corruption issues:

- `CONFIG_PAGE_POISONING`
- `CONFIG_SLUB_DEBUG_ON`
- `CONFIG_DEBUG_LIST`
- `CONFIG_SLUB_DEBUG`

Sample log:

```
BUG <slab cache affected>: <What went wrong>
---------------------------------------------
INFO: <corruption start>-<corruption_end> <more info>
INFO: Slab <address> <slab information>
INFO: Object <address> <object information>
INFO: Allocated in <kernel function> age=<jiffies since alloc> cpu=<allocated by cpu>
pid=<pid of the process>
INFO: Freed in <kernel function> age=<jiffies since free> cpu=<freed by cpu> pid=<pid of
the process>
```

For more information about slub debugging, see `Documentation/vm/slub.txt` available at kernel documentation.

Out of memory

When the system fails to assign a page, the kernel logs display a message such as the following:

```
<4>[12146.861355] Thread-430: page allocation failure: order:0, mode:0x10d2
<CALL STACK>
<4>[12146.951687] Mem-info:
<4>[12146.953909] Normal per-cpu:
<4>[12146.956686] CPU 0: hi: 186, btch: 31 usd: 61
<4>[12146.961489] CPU 1: hi: 186, btch: 31 usd: 0
<4>[12146.966235] HighMem per-cpu:
<4>[12146.969122] CPU 0: hi: 186, btch: 31 usd: 54
<4>[12146.973877] CPU 1: hi: 186, btch: 31 usd: 0
.......
<4>[12147.010770] Normal free:53192kB min:3508kB low:4384kB high:5260kB ........
<4>[12147.050805] lowmem_reserve[]: 0 9022 9022
<4>[12147.054610] HighMem free:153360kB min:512kB low:1824kB high:3140kB .........
<4>[12147.095453] lowmem_reserve[]: 0 0 0
<4>[12147.098617] Normal: 118*4kB 232*8kB 161*16kB 110*32kB 34*64kB 9*128kB
10*256kB 6*512kB 7*1024kB 6*2048kB 4*4096kB = 53224kB
<4>[12147.115455] HighMem: 2774*4kB 11769*8kB 3005*16kB 1*32kB 0*64kB 0*128kB
0*256kB 0*512kB 0*1024kB 0*2048kB 0*4096kB = 153360kB
..................
```

These messages indicate that the system couldn't assign the requested page. The top line of the log message provides several details about the out-of-memory issue. For example, in the following log:

```
<4>[1214.855361] Thread-4: page allocation failure: order:2, mode:0x10d2
```

- **order**: Indicates the size of the page. In this example, 2^2 x PAGE_SIZE (4 K) = 16 K
  - Linux uses a buddy allocator that allocates pages in powers of 2.
    - The maximum size of the buddy allocator is the order of 10 = 2^{10} x 4 kB = 4 MB.
    - For allocation > 4 MB, use an alternate allocation method such as the contiguous memory allocator (CMA).
  - For failure of higher-order allocations, examine whether the memory can be virtually contiguous, instead of being physically contiguous.
- **mode**: Indicates the type of page requested
  - `mode` provides information about get free pages (GFP) flags. In this example, `mode` is

`0x10d2`.

– `mode` is the result of OR operation on all the GFP flags in the allocation.

- Pages available in the system

  A page allocation failure message prints details about the size of pages that were available in the system.

```
Normal: 118*4kB 232*8kB 161*16kB 110*32kB 34*64kB 9*128kB 10*256kB 6*512kB
7*1024kB 6*2048kB 4*4096kB = 53224kB
HighMem: 2774*4kB 11769*8kB 3005*16kB 1*32kB 0*64kB 0*128kB 0*256kB 0*512kB
0*1024kB 0*2048kB 0*4096kB = 153360kB
```

IOMMU page fault

IOMMU, also known as the system MMU (SMMU), performs memory management functions on behalf of subsystems that don't have their own MMU.

The IOMMU hardware block allows physically noncontiguous pages to support virtually contiguous memory. The memory translation logic in the IOMMU is the same as the logic in the CPU MMU.

The IOMMU page fault is the most commonly seen IOMMU issue. The fault occurs when the requested page is mapped in the page table but isn't found in the memory. The fault handler receives the context bank instance of the IOMMU and dumps the registers for this context.

The following log indicates an IOMMU page fault.

```
[ 47.228992] msm_iommu_v1: Unexpected IOMMU page fault!
[ 47.233115] msm_iommu_v1: name = mdp_iommu
[ 47.237238] msm_iommu_v1: context = mdp_0 (0)
[ 47.241507] msm_iommu_v1: Interesting registers:
[ 47.246149] msm_iommu_v1: FAR = 0000000000000000
[ 47.250970] msm_iommu_v1: PAR = 0000000000000000
[ 47.255834] msm_iommu_v1: FSR = 00000002 [TF ]
[ 47.260540] msm_iommu_v1: FSYNR0 = 000005a1 FSYNR1 = 00030005
[ 47.266528] msm_iommu_v1: TTBR0 = 0000000071a28000
[ 47.271370] msm_iommu_v1: TTBR1 = 0000000000000000
[ 47.276248] msm_iommu_v1: SCTLR = 00001043 ACTLR = 70000000
[ 47.282221] msm_iommu_v1: CBAR = 00000000 CBFRSYNRA = 00000000
[ 47.288521] msm_iommu_v1: PRRR = ff0a81a8 NMRR = 40e040e0
[ 47.294461] msm_iommu_v1: NOTE: Value actually unknown for CBAR
[ 47.300394] msm_iommu_v1: NOTE: Value actually unknown for CBFRSYNRA
[ 47.306717] msm_iommu_v1: Page table in DDR shows PA = 0
```

The following table describes the fields captured in the log message.

Table : Information in IOMMU page fault log

Item	Description
`name`	Name of the hardware block that caused the fault.
`FAR`	Fault address register (FAR) indicates the address at which the fault occurred.
`FSR`	Fault status register (FSR) indicates the following: • Translation fault (TF) • Access permission fault (APF) • Stalled status (SS)

The FSR is one of the most important registers in IOMMU debugging. This register has read/write-clear access. The read operation on this register reads the value in the register while the write operation clears the bits corresponding to 1s in the written data and leaves the bits corresponding to 0s unchanged. This process prevents inadvertent clearing of new faults when writing the register to clear an old fault. Some useful bits in this register are:

Table : Bits in fault status register

Bit	Description
`[Bit 1]: TF`	Translation fault (invalid page table entry)
`[Bit 2]: AFF`	Access fault
`[Bit 3]: APF`	Permission fault (write to read only region)
`[Bit 4]: TLBMF`	TLB miss fault
`[Bit 5]: HTWDEEF`	Hardware table walk decode error external fault
`[Bit 6]: HTWSEEF`	Hardware table walk subordinate error external fault
`[Bit 7]: MHF`	Many hits in TLB
`[Bit 16]: SL`	Second-level fault (fault occurred in second level of page table)
`[Bit 30]: SS`	Stalled status
`[Bit 31]: MULTI`	Multiple faults

The `TF`, `APF`, and `SL` flags indicate normal operation, whereas `TLBMF`, `HTWDEEF`, `HTWSEEF`, and `MHF` flags indicate that there is an issue.

IOMMU page table

The IOMMU page table dump provides a faulting address from the FAR and the register dump in the kernel log. This faulting address represents the virtual address, and you can acquire the corresponding physical address from the page table. From the page table dump, you can identify whether the requested address is mapped or not mapped. Each IOMMU domain has a page table, and the dump includes page tables for each of the domains. Currently, there are six domains.

The following is the sample dump of the `Domain: 2` page table.

```
Domain: 2 [L2 cache redirect for page tables is OFF]
0x00000000--0x0001ffff [0x00020000] [UNMAPPED]
0x00020000--0x01807fff [0x017e8000] A:0x82a8e000--0x84275fff [0x017e8000] [R/W][4K]
0x01808000--0x01939fff [0x00132000] A:0xf0c24000--0xf0d55fff [0x00132000] [R/W][4K]
0x0193a000--0x0199ffff [0x00066000] A:0xf13fa000--0xf145ffff [0x00066000] [R/W][4K]
0x019a0000--0x01e85fff [0x004e6000] A:0xf966e000--0xf9b53fff [0x004e6000] [R/W][4K]
0x01e86000--0x01ffffff [0x0017a000] [UNMAPPED]
0x02000000--0x02feffff [0x00ff0000] A:0xf5a22000--0xf6a11fff [0x00ff0000] [R/W][4K]
```

In this example, the first column represents the virtual address, the second column represents the number of bytes in the corresponding region of contiguous physical addresses, and the third column represents the physical addresses. The permissions are also mentioned for each of these regions.

Memory map

For more information about the memory map, see the latest Release Notes.

Stack corruption

Wrong coding logic accesses memory locations in the stack, leading to changes in values at those memory locations, causing stack corruption. Stack corruption can occur in the following ways:

- Due to bad code logic, the program consumes all the stack memory, and it writes memory beyond the stack boundaries, causing a stack overflow.
- Accessing an array that's out of bounds.
- An undefined or freed pointer that points at a stack address.
- Corrupted return address of a caller function.

To identify the stack corruption issues, enable the following kernel configuration options:

- `CONFIG_STACKPROTECTOR`
- `CONFIG_STACKPROTECTOR_STRONG`

The Kernel address sanitizer (KASAN) utility also helps in identifying some stack corruption issues.

3.6 Function tracer

Function tracer (ftrace) provides tracing utilities to perform system-wide profiling and tracing at runtime.

To use ftrace, enable the following configuration options:

- `CONFIG_FTRACE`
- `CONFIG_HAVE_FUNCTION_TRACER`
- `CONFIG_HAVE_FUNCTION_GRAPH_TRACER`
- `CONFIG_HAVE_DYNAMIC_FTRACE`
- `CONFIG_HAVE_FTRACE_MCOUNT_RECORD`

The following are some operations that ftrace can perform to debug kernel issues:

Dump ftrace information to kmsg buffer

To dump the ftrace information into the kmsg buffer anytime from the source code, call the `ftrace_dump(DUMP_ALL)` function.

To increase the buffer size of the ftrace ring, run the following command:

```
echo 200 > /sys/kernel/debug/tracing/buffer_size_kb
```

Enable work queue trace

To enable work queue tracing, run the following commands:

```
mount -t debugfs none /sys/kernel/debug
```

```
echo 1 > /sys/kernel/debug/tracing/events/workqueue/enable
```

```
echo workqueue:workqueue_queue_work > /sys/kernel/debug/tracing/set_
event
```

```
cat /sys/kernel/debug/tracing/trace_pipe
```

```
cat /sys/kernel/debug/tracing/per_cpu/cpu1/trace
```

Sample output:

```
# tracer: nop
#
# entries-in-buffer/entries-written: 8682/8682 #P:1
#
# _-----=> irqs-off
# / _----=> need-resched
# | / _---=> hardirq/softirq
# || / _--=> preempt-depth
# ||| / delay
# TASK-PID CPU# |||| TIMESTAMP FUNCTION
# | | | |||| | |
        <...>-4783 [001] d.s4 7524.354249: workqueue_queue_work: work struct=f2d91ee4
        function=free_css_set_work workqueue=f6427d80 req_cpu=1 cpu=1
        <idle>-0 [001] d.h4 7524.424196: workqueue_queue_work: work struct=c10c32a8
        function=def_work_fn workqueue=f55c7880 req_cpu=1 cpu=4
e.process.gapps-4758 [001] dNs4 7524.454227: workqueue_queue_work: work
struct=c4727fa4 function=free_css_set_work workqueue=f6427d80 req_cpu=1 cpu=1
        Binder_D-1693 [001] d.s3 7524.504198: workqueue_queue_work: work
        struct=cec6275c function=do_dbs_timer workqueue=f5424680 req_cpu=1 cpu=1
                <...>-4832 [001] d.h3 7524.574194: workqueue_queue_work: work
                struct=c10c32a8 function=def_work_fn workqueue=f55c7880 req_cpu=1 cpu=4
```

Enable Interrupt request (IRQ) trace

To enable interrupt tracing, run the following commands:

```
mount -t debugfs none /sys/kernel/debug
```

```
echo 1 > /sys/kernel/debug/tracing/events/irq/irq_handler_entry/
enable
```

```
cat /sys/kernel/debug/tracing/trace
```

```
cat /sys/kernel/debug/tracing/trace_pipe
```

Sample output:

```
TASK-PID CPU# TIMESTAMP FUNCTION
          | | | | |
     adbd-302 [000] 295.075568: irq_handler_entry: irq=132 name=msm_otg
     adbd-302 [000] 295.075599: irq_handler_entry: irq=132 name=msm_hsusb
     adbd-302 [000] 295.075782: irq_handler_entry: irq=132 name=msm_otg
     adbd-302 [000] 295.075782: irq_handler_entry: irq=132 name=msm_hsusb
<idle>-0 [000] 295.076270: irq_handler_entry: irq=132 name=msm_otg
<idle>-0 [000] 295.076270: irq_handler_entry: irq=132 name=msm_hsusb
<idle>-0 [000] 295.077155: irq_handler_entry: irq=18 name=gp_timer
<idle>-0 [000] 295.087166: irq_handler_entry: irq=18 name=gp_timer
<idle>-0 [000] 295.097146: irq_handler_entry: irq=18 name=gp_timer
```

Enable scheduler trace

To trace context switches between tasks, run the following commands to enable the `sched_switch` tracer:

```
mount -t debugfs none /sys/kernel/debug
```

```
echo 1 > /sys/kernel/debug/tracing/events/sched/sched_switch/enable
```

```
cat /sys/kernel/debug/tracing/trace
```

```
cat /sys/kernel/debug/tracing/trace_pipe
```

Sample output:

```
TASK-PID CPU# TIMESTAMP FUNCTION
        | | | | |
WindowManagerPo-484 [000] 994.583135: sched_switch: prev_comm=WindowManagerPo
SurfaceFlinger-334 [000] 994.583652: sched_switch: prev_comm=SurfaceFlinger
WindowManagerPo-484 [000] 994.584320: sched_switch: prev_comm=WindowManagerPo
GL updater-675 [000] 994.584370: sched_switch: prev_comm=GL updater
WindowManagerPo-484 [000] 994.584424: sched_switch: prev_comm=WindowManagerPo
```

Find IRQ latency

To find the maximum IRQ latency and pre-emption latency in the system, enable ftrace configurations for `IRQOFF` and `PREEMPTIONOFF` as follows:

```
CONFIG_IRQSOFF_TRACER =Y
CONFIG_PREEMPT_TRACER =Y
```

For this configuration to take effect, recompile the kernel. This configuration detects latency in milliseconds effectively.

To enable tracing, run the following commands:

```
echo 0 > /sys/kernel/debug/tracing/tracing_enabled
```

```
echo 0 > /sys/kernel/debug/tracing/tracing_max_latency
```

```
echo irqsoff > /sys/kernel/debug/tracing/current_tracer
```

```
echo 1 > /sys/kernel/debug/tracing/tracing_enabled
```

```
cat /sys/kernel/debug/tracing/trace
```

To find the maximum latency observed in the system, configure `tracing_max_latency = 0`. To detect any latency higher than the specified limit, set the threshold level in microseconds. For example,

```
echo 2000 > /sys/kernel/debug/tracing/tracing_thresh
```

The following example shows the traces for IRQ latency of 16 ms:

```
cat /sys/kernel/debug/tracing/trace

# tracer: irqsoff
#
# WARNING: FUNCTION TRACING IS CORRUPTED
# MAY BE MISSING FUNCTION EVENTS
# irqsoff latency trace v1.1.5 on 3.4.0-perf-g7736d93-dirty
# --------------------------------------------------------------------
# latency: 16757 us, #4/4, CPU#0 | (M:preempt VP:0, KP:0, SP:0 HP:0 #P:1)
# -----------------
# | task: EventThread-534 (uid:1000 nice:-9 policy:0 rt_prio:0)
# -----------------
# => started at: _raw_spin_lock_irqsave
# => ended at: _raw_spin_unlock_irqrestore
#
#
# _------=> CPU#
# / _-----=> irqs-off
# | / _----=> need-resched
# || / _---=> hardirq/softirq
# ||| / _--=> preempt-depth
# |||| / delay
# cmd pid ||||| time | caller
# / ||||| | /
        <...>-534 0d... 0us!: _raw_spin_lock_irqsave
        <...>-534 0d..1 16756us+: _raw_spin_unlock_irqrestore
        <...>-534 0d..1 16758us+: trace_hardirqs_on <-_raw_spin_unlock_irqrestore
        <...>-534 0d..1 16780us : <stack trace>
```

```
=> trace_hardirqs_on
=> _raw_spin_unlock_irqrestore
=> clk_enable
=> mdss_dsi_clk_enable
=> mdss_dsi_clk_ctrl
=> mdss_dsi_clk_req
=> mdss_dsi_event_handler
=> mdss_mdp_ctl_intf_event
=> mdss_mdp_cmd_add_vsync_handler
=> mdss_mdp_overlay_vsync_ctrl
=> mdss_mdp_overlay_ioctl_handler
=> mdss_fb_ioctl
=> do_fb_ioctl
=> fb_ioctl
=> do_vfs_ioctl
=> sys_ioctl
=> ret_fast_syscall
/sys/kernel/debug/tracing #
```

For more information, see the `Documentation/trace/ftrace.txt` file available at kernel documentation.

3.7 Collect and parse RAM dump

A RAM dump is a snapshot of the system memory at the time of device failure and is useful for debugging various crash issues.

Enable RAM dump

By default, the `debug` build incorporates the RAM dump. However, it's disabled in the `perf/non-DEBUG` build using the kernel command-line parameter. To configure the RAM dump, use the `qcom_scm.download_mode` parameter in the `meta-qcom-hwe/conf/machine/include/<qcom-qcs6490.inc>` file as:

- 0: Disable
- 1: Enable

The following table lists the `.inc` file corresponding to different chipsets:

Chipset	Filename
QCS5430/QCS6490	`meta-qcom-hwe/conf/machine/include/qcom-qcs6490.inc`
QCS8275	`meta-qcom-hwe/conf/machine/include/qcom-qcs8300.inc`
QCS9075	`meta-qcom-hwe/conf/machine/include/qcom-qcs9100.inc`
QCS615	`meta-qcom-hwe/conf/machine/include/qcom-qcs615.inc`

Collect RAM dump

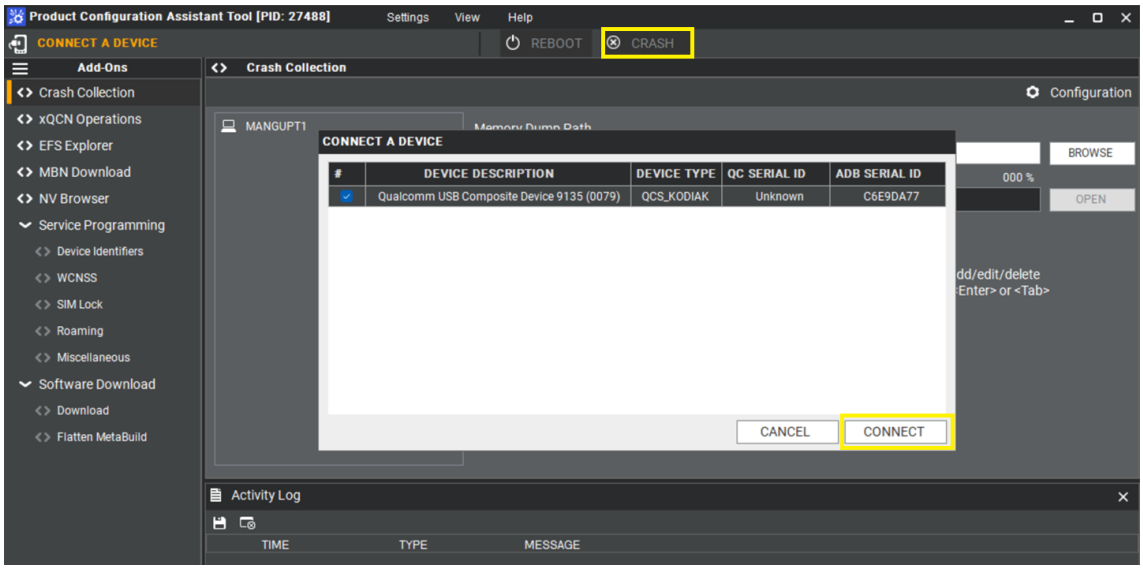
To collect the RAM dump over USB, use the Product Configuration Assistant Tool (PCAT). Download the PCAT tool from the Qualcomm Software Center. After installing PCAT, access the PCAT user guide at the following locations:

- Windows host: `C:\Program Files(x86)\Qualcomm\PCAT\Docs`
- Linux host: `/opt/qcom/PCAT/Docs/`

To collect RAM dump using PCAT, do the following:

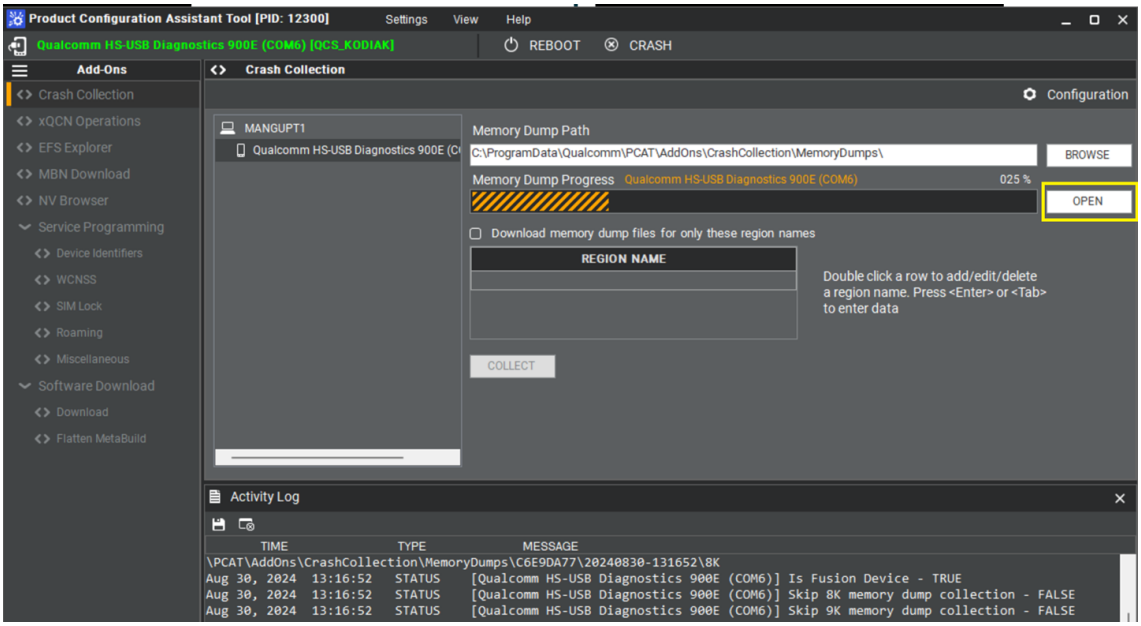
1. Start PCAT on a Windows host and connect the device, which is already in the RAM dump mode.

   If necessary, to trigger a device crash, select **CRASH** on the PCAT interface.

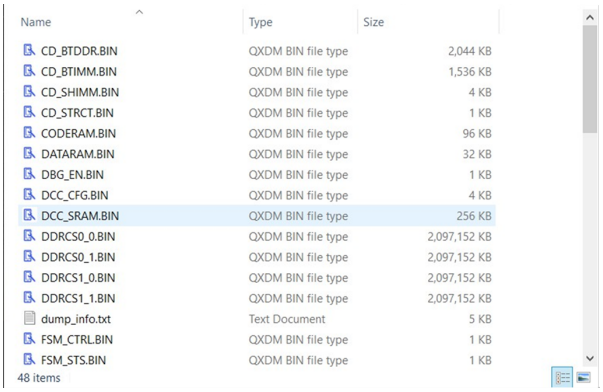


   After you connect the device to the Windows host, PCAT automatically starts capturing the RAM dump, and the PCAT UI displays the progress of the dump collection.

2. To view the dump, select **OPEN**.



The following is an example showing the contents of a RAM dump directory:



Name	Type	Size
CD_BTDDR.BIN	QXDM BIN file type	2,044 KB
CD_BTIMM.BIN	QXDM BIN file type	1,536 KB
CD_SHIMM.BIN	QXDM BIN file type	4 KB
CD_STRCT.BIN	QXDM BIN file type	1 KB
CODERAM.BIN	QXDM BIN file type	96 KB
DATARAM.BIN	QXDM BIN file type	32 KB
DBG_EN.BIN	QXDM BIN file type	1 KB
DCC_CFG.BIN	QXDM BIN file type	4 KB
DCC_SRAM.BIN	QXDM BIN file type	256 KB
DDRCS0_0.BIN	QXDM BIN file type	2,097,152 KB
DDRCS0_1.BIN	QXDM BIN file type	2,097,152 KB
DDRCS1_0.BIN	QXDM BIN file type	2,097,152 KB
DDRCS1_1.BIN	QXDM BIN file type	2,097,152 KB
dump_info.txt	Text Document	5 KB
FSM_CTRL.BIN	QXDM BIN file type	1 KB
FSM_STS.BIN	QXDM BIN file type	1 KB

48 items

Note: The Linux host uses a command-line interface, instead of the GUI for the PCAT-based RAM dump capture, such as `PCAT -PLUGIN CC -DEVICE <serial-id> -DUMPDIR /tmp -RESET TRUE -UNIQUETS TRUE`.

3.8 Parse RAM dump using RAMParser

Linux RAM dump parser (RAMParser) is an open-source tool used to parse RAM dump on Qualcomm Linux devices. The RAMParser processes the RAM dump using the Linux kernel symbol file, which includes vmlinux and kernel object modules, and extracts useful information such as process stacks, IRQ, and workqueues.

Note: The RAMParser tool is built and verified only for Windows.

Prerequisites

- RAM dump and the corresponding `vmlinux` file
- Software images and scripts
  - Windows PC
  - Python 3.7 or a later version

```
python -m pip install --trusted-host files.pythonhosted.org --
trusted-host pypi.org --trusted-host pypi.python.org
prettytable
```

```
python -m pip install --trusted-host files.pythonhosted.org --
trusted-host pypi.org --trusted-host pypi.python.org
pyelftools
```

- Pyelftools package
  1. Download Pyelftools package from https://github.com/eliben/pyelftools.
  2. Unzip the downloaded file and look for `pyelftools-master>elftools` directory.
  3. Copy elftools directory to `<installed Python path>\Lib\site-packages`.
- RAMParser software

  Download the RAMParser software from https://git.codelinaro.org/clo/la/platform/vendor/qcom-opensource/tools/-/tree/opensource-tools.lnx.1.0.

  Note: The RAMParser software must always be present in the C drive on a Windows host.

- (Optional) TRACE32 software 2023.12 or a later version on C drive (`C:\T32`)

  The TRACE32 software loads the RAM dump in the TRACE32 simulator using the RAMParser output files.

Set up toolchains

The RAMParser requires access to gdb and nm tools. You can specify the paths to the gdb and nm tools in one of the following ways:

- Using `--gdb-path` and `--nm-path` to specify the absolute path
- Using `CROSS_COMPILE` to specify the prefix
- Using `local_settings.py` file

Note: Availability of gdb or nm only on the path isn't supported because there are too many variations on the names to invoke.

To set up toolchains using the `local_settings.py` file, do the following:

1. Create a directory named `ramparser_utils\utils` in the root directory.
2. Download the toolchain (*aarch64-none-linux-gnu*) in the `ramparser_utils\utils` directory from https://developer.arm.com/downloads/-/arm-gnu-toolchain-downloads.

local_settings.py

The RAMParser automatically determines most of the settings. However, some settings are unique to the runtime environment. You specify these unique settings in the `local_settings.py` file. As `local_settings.py` is a Python file, it can leverage Python features.

Note: For RAMParser to pick the correct path for toolchain utilities, add the `local_settings.py` file to the path `<root>/tools/linux-ramdump-parser-v2` directory.

The format of the `local_settings.py` file is:

```
<setting name> = <string identifying the feature>
```

The `local_settings.py` file supports the following features:

Table : Supported features in local_settings.py file

Feature	Description
gdb_path	Absolute path to the gdb tool for the RAM dump
nm_path	Absolute path to the nm tool for the RAM dump
gdb64_path	Absolute path to the 64-bit gdb tool for the RAM dump
nm64_path	Absolute path to the 64-bit nm tool for the RAM dump
objdump_path	Absolute path to the object dump tool for the RAM dump

Feature	Description
objdump64_path	Absolute path to the 64-bit object dump tool for the RAM dump

Example: local_settings.py

```
import parser_util,os,sys
path = os.path.abspath(os.path.dirname(__file__))
ramparser_utils_path = os.path.abspath(os.path.join(path, "../../
ramparser_utils"))
nm_path =  ramparser_utils_path + "\\utils\\arm-none-eabi-nm.exe"
gdb_path = ramparser_utils_path + "\\utils\\arm-none-eabi-gdb.exe"
objdump_path =  ramparser_utils_path + "\\utils\\objdump.exe"
nm64_path = ramparser_utils_path + "\\utils\\aarch64-linux-gnu-nm.exe
"
gdb64_path = ramparser_utils_path + "\\utils\\mingw64\\bin\\gdb-
multiarch.exe"
objdump64_path = ramparser_utils_path + "\\utils\\all-objdump.exe"
```

RAMParser commands

To parse dumps using RAMParser, run the following command in the Windows Shell:

```
python ramparse.py --vmlinux <vmlinux path> --auto-dump <dump path> -
-force-hardware <hw name> <parser options>  --mod_path <symbol path>
-o <output path>
```

- `<hw name>`: This string specifies the hardware ID of the chipset. See the following table and use the appropriate value.

Chipset	Hardware name
QCS6490	qcm6490
QCS5430	
QCS8275	qcs8300
QCS9075	qcs9100
QCS615	qcs615

- `python ramparse.py`: This file invokes the RAMParser.
- `mod_path`: Specify this option to copy all the unstripped kernel modules into a directory and reference it as the symbol path for the `--mod_path` option.
- `<parser options>`: Specify parser options to extract specific data from the RAM dump.

Additionally, you can pass subparser options in the command to extract the relevant data. For example, to extract ftrace information, pass `--dump-ftrace` along with the related arguments and run the command:

```
python <root>\tools\linux-ramdump-parser-v2\ramparse.py --
vmlinux <vmlinux path> --auto-dump <dump path> --force-hardware
<hw name> --dump-ftrace --ftrace-args=rwmmio --ftrace_buffer_
size_kb 4096  --mod_path <symbol path> -o <output path>
```

Similarly, to extract the kconfig information, pass `--print-kconfig`. There are many options available to parse the dumps and store the output in a directory.

To view all the available commands and options, run the following command:

```
python ramparse.py --help
```

RAMParser logs

The RAMParser generates an extensive amount of data regarding work queues, processes states, and call stacks. The following table lists the important files generated during RAMParser execution.

Table : Files generated by RAMParser

Filename	Description
`dmesg_TZ.txt`	This file includes kernel logs, run queues, work queues, and IRQ statistics.
`mem_stat.txt`	This file provides statistics about system memory.
`memory.txt`	
`tasks.txt`	This file provides a kernel space call stack for all processes.
`devicetree.dtb`	This file is the device tree blob used by the kernel.
`launch_t32.bat`	This file launches the TRACE32 simulator launcher file for loading RAM dumps.
`timerlist.txt`	This file provides the list of active timers.

The following screenshots show sample data generated by the RAMParser:

```
======================= RUNQUEUE STATE ============================
CPU0 1 process is running
curr: kworker/0:0     (     8) [affinity=0x01] [vruntime=     11964851837]
idle: swapper/0       (     0) [affinity=0x01] [vruntime=               0]
stop: migration/0     (    17) [affinity=0x01] [vruntime=        25875544]
CFS 1 process is pending
curr: kworker/0:0     (     8) [affinity=0x01] [vruntime=     11964851837]
 |--next: None(0)
RT 0 process is pending
current callstack is maybe:
0xffff800080092fe8:('vsnprintf', 932)
0xffff800080093068:('sprintf', 100)
0xffff800080093118:('__sprint_symbol.constprop.0', 260)
0xffff800080093188:('sprint_symbol_build_id', 24)
0xffff8000800931a8:('number', 840)
0xffff8000800931d8:('vsnprintf', 932)
0xffff800080093208:('vsnprintf', 348)
0xffff800080093258:('sprintf', 100)
0xffff800080093308:('info_print_prefix', 136)
0xffff800080093328:('_prb_read_valid', 156)
0xffff800080093388:('prb_read_valid', 28)
0xffff8000800933e8:('printk_get_next_message', 116)
0xffff800080093418:('__qcom_geni_serial_console_write', 208)
0xffff800080093428:('__qcom_geni_serial_console_write', 340)
0xffff800080093448:('qcom_geni_serial_console_write', 400)
0xffff800080093498:('console_flush_all.constprop.0', 264)
0xffff8000800934d8:('_prb_read_valid', 156)
0xffff800080093538:('prb_read_valid', 28)
0xffff800080093580:('panic_handler', 0)
0xffff800080093598:('console_unlock', 116)
0xffff8000800935d8:('__wake_up_klogd.part.0', 156)
0xffff8000800935f8:('vprintk_emit', 480)
0xffff800080093618:('dev_printk_emit', 148)
0xffff800080093738:('__dev_printk', 60)
0xffff800080093788:('_dev_err', 100)
0xffff8000800937d8:('gh_show_wdt_status', 148)
0xffff800080093818:('qcom_wdt_trigger_bite', 96)
0xffff800080093838:('panic_handler', 28)
0xffff800080093868:('notifier_call_chain', 116)
0xffff800080093878:('atomic_notifier_call_chain', 60)
0xffff8000800938d8:('panic', 388)
0xffff800080093958:('die', 556)
0xffff800080093960:('fw_devlink_no_driver', 8)
0xffff800080093998:('oops_exit', 52)
0xffff8000800939a0:('fw_devlink_no_driver', 8)
0xffff800080093a38:('die_kernel_fault', 432)
0xffff800080093a98:('do_alignment_fault', 0)
0xffff800080093ae8:('do_translation_fault', 80)
0xffff800080093b18:('do_mem_abort', 68)
0xffff800080093b38:('el1_abort', 64)
```

Figure : Runqueue (dmesg_TZ.txt)

```
----------begin IrqParse----------
IRQ  HWIRQ        affinity   CPU0       CPU1       CPU2       CPU3       CPU4       CPU5       CPU6       CPU7       Name                       Chip          IRQ Structure
   1 0x0          0xffffffff 772        3088       713        754        1278       645        650        1650       IPI                        GICv3         v.v (struct irq_desc *)0xffff700c80012200
   2 0x1          0xffffffff 5265       10669      6642       6960       6681       4130       5338       7282       IPI                        GICv3         v.v (struct irq_desc *)0xffff700c80012400
   3 0x2          0xffffffff 0          1          1          1          1          1          1          1          IPI                        GICv3         v.v (struct irq_desc *)0xffff700c80012600
   4 0x3          0xffffffff 0          0          0          0          0          0          0          0          IPI                        GICv3         v.v (struct irq_desc *)0xffff700c80012800
   5 0x4          0xffffffff 185        490        711        601        555        751        730        470        IPI                        GICv3         v.v (struct irq_desc *)0xffff700c80012a00
   6 0x5          0xffffffff 1260       570        1051       850        570        447        344        1181       IPI                        GICv3         v.v (struct irq_desc *)0xffff700c80012c00
   7 0x6          0xffffffff 0          0          0          0          0          0          0          0          IPI                        GICv3         v.v (struct irq_desc *)0xffff700c80012e00
   9 0x26         0x10       412        406        253        256        755        564        416        1031       arch_mem_timer             GICv3         v.v (struct irq_desc *)0xffff700c80013400
  13 0x1b         0xffffffff 5301       6303       5625       4945       2940       2398       2243       5102       arch_timer                 GICv3         v.v (struct irq_desc *)0xffff700c80013c00
  15 0x105        0xff       5476       0          0          0          0          0          0          0          ipcc_0                     GICv3         v.v (struct irq_desc *)0xffff700c80c7f200
  16 0x25         0xff       7389       0          0          0          0          0          0          0          18200000.rsc               GICv3         v.v (struct irq_desc *)0xffff700c80c7ce00
  17 0x20         0xff       0          0          0          0          0          0          0          0          apps_wdog_bark             GICv3         v.v (struct irq_desc *)0xffff700c80c7e400
  18 0x0          0xffffffff 401        0          0          0          0          0          0          0          aoss-qmp                   ipcc          v.v (struct irq_desc *)0xffff700c829c6000
  19 0x30002      0xffffffff 3          0          0          0          0          0          0          0          smp2p                      ipcc          v.v (struct irq_desc *)0xffff700c829c4000
  20 0x60002      0xffffffff 3          0          0          0          0          0          0          0          smp2p                      ipcc          v.v (struct irq_desc *)0xffff700c829c4a00
  21 0x20002      0xffffffff 0          0          0          0          0          0          0          0          smp2p                      ipcc          v.v (struct irq_desc *)0xffff700c829c6c00
  22 0x180002     0xffffffff 4          0          0          0          0          0          0          0          smp2p                      ipcc          v.v (struct irq_desc *)0xffff700c829c5400
  23 0x2c1        0xff       0          0          0          0          0          0          0          0          arm-smmu global fault      GICv3         v.v (struct irq_desc *)0xffff700c80debc00
  24 0x2c3        0xff       0          0          0          0          0          0          0          0          arm-smmu global fault      GICv3         v.v (struct irq_desc *)0xffff700c80de5600
  25 0x2c6        0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80deb000
  26 0x2c7        0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80deba00
  27 0x2c8        0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80de9200
  35 0x61         0xff       0          0          0          0          0          0          0          0          arm-smmu global fault      GICv3         v.v (struct irq_desc *)0xffff700c80deb600
  36 0x80         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80dea600
  37 0x81         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80de8800
  38 0x82         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80df2e00
  39 0x83         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80df1e00
  40 0x84         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80df2800
  41 0x85         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80df1c00
  42 0x86         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80df3200
  43 0x87         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80df2a00
  44 0x88         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80df0e00
  45 0x89         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80df2400
  46 0x8a         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80df3e00
  47 0x8b         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80df0400
  48 0x8c         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80df1000
  49 0x8d         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80df1800
  50 0x8e         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80df3c00
  51 0x8f         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80df0600
  52 0x90         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80df3000
  53 0x91         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80df3a00
  54 0x92         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80df1200
  55 0x93         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80df0200
  56 0x94         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80df2000
  57 0x95         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80df3400
  58 0x96         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80df0000
  59 0xd5         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80df0a00
  60 0xd6         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80df2c00
  61 0xd7         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80df1400
  62 0xd8         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80df1600
  63 0xd9         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80df3800
  64 0xda         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80df2200
  65 0xdb         0xff       0          0          0          0          0          0          0          0          arm-smmu-context-fault     GICv3         v.v (struct irq_desc *)0xffff700c80df3600
```

Figure : IRQ state (dmesg_TZ.txt)

```
[   30.803927][    T8] Unable to handle kernel paging request at virtual address ffff700c000002f0
[   30.812205][    T8] Mem abort info:
[   30.815100][    T8]   ESR = 0x0000000096000006
[   30.819070][    T8]   EC = 0x25: DABT (current EL), IL = 32 bits
[   30.824610][    T8]   SET = 0, FnV = 0
[   30.827828][    T8]   EA = 0, S1PTW = 0
[   30.831065][    T8]   FSC = 0x06: level 2 translation fault
[   30.836136][    T8] Data abort info:
[   30.839103][    T8]   ISV = 0, ISS = 0x00000006, ISS2 = 0x00000000
[   30.844799][    T8]   CM = 0, WnR = 0, TnD = 0, TagAccess = 0
[   30.850036][    T8]   GCS = 0, Overlay = 0, DirtyBit = 0, Xs = 0
[   30.855532][    T8] swapper pgtable: 4k pages, 48-bit VAs, pgdp=0000000233da0000
[   30.862481][    T8] [ffff700c000002f0] pgd=180000027fff7003, p4d=180000027fff7003, pud=180000027
[   30.873437][    T8] Internal error: Oops: 0000000096000006 [#1] PREEMPT SMP
[   30.879872][    T8] Modules linked in: usb_f_uvc videobuf2_vmalloc uvc videobuf2_dma_sg videobuf
[   30.880021][    T8]  snd_soc_qcom_common(E) videodev(E) nvmem_reboot_mode(E) pmic_glink(E) mc(E)
[   31.020116][    T8] CPU: 0 PID: 8 Comm: kworker/0:0 Tainted: G        W  OE      6.6.0 #1
[   31.027802][    T8] Hardware name: Qualcomm Technologies, Inc. qcm6490-addons IDP platform (DT)
[   31.036013][    T8] Workqueue: events deferred_probe_timeout_work_func
[   31.042016][    T8] pstate: 40400005 (nZcv daif +PAN -UAO -TCO -DIT -SSBS BTYPE=--)
[   31.049165][    T8] pc : fw_devlink_no_driver+0x4/0x64
[   31.053731][    T8] lr : class_for_each_device+0x114/0x15c
[   31.058657][    T8] sp : ffff800080093cf0
[   31.062064][    T8] x29: ffff800080093d20 x28: ffffacclee3d79c0 x27: ffffacclee3d7000
[   31.069389][    T8] x26: ffff700c8000ad00 x25: ffff700c8000ad40 x24: ffff700c80c7f800
[   31.076714][    T8] x23: ffff700c80c7f800 x22: ffffaccled03419c x21: 0000000000000000
[   31.084040][    T8] x20: 0000000000000000 x19: ffff800080093cf8 x18: 0000000000000000
[   31.091365][    T8] x17: 0000000000000000 x16: ffffaccled374708 x15: 0000000000000036
[   31.098692][    T8] x14: 0000000000000000 x13: 0000000000000030 x12: 0101010101010101
[   31.106018][    T8] x11: 7f7f7f7f7f7f7f7f x10: ffff700c8551b9c0 x9 : 0000000000000001
[   31.113342][    T8] x8 : ffffffffffffffff x7 : 0000000000000162 x6 : 000000000002888d
[   31.120667][    T8] x5 : 0000000000000001 x4 : ffff700c801ee180 x3 : ffff700c80953d80
[   31.127994][    T8] x2 : 0000000000000001 x1 : ffff700c00000000 x0 : ffff700c82f6fc30
[   31.135320][    T8] Call trace:
[   31.137836][    T8]  fw_devlink_no_driver+0x4/0x64
[   31.142049][    T8]  fw_devlink_drivers_done+0x48/0x60
[   31.146611][    T8]  deferred_probe_timeout_work_func+0x18/0xa0
[   31.151977][    T8]  process_one_work+0x160/0x3a8
[   31.156102][    T8]  worker_thread+0x32c/0x438
[   31.159960][    T8]  kthread+0x118/0x11c
[   31.163281][    T8]  ret_from_fork+0x10/0x20
[   31.166958][    T8] Code: a8c37bfd d50323bf d65f03c0 f85d0001 (f9417821)
[   31.173214][    T8] ---[ end trace 0000000000000000 ]---
[   31.177963][    T8] Kernel panic - not syncing: Oops: Fatal exception
```

Figure : Sample kernel crash (dmesg_TZ.txt)

```
Timer List Dump

------------------------------------------------------------------------------------------------------------------------
CPU 0(tvec_base: ffff700dff0825c0 timer_jiffies: 4294899985(30.76s) next_timer: 4294900032(30.948s) active_timers: NA)
------------------------------------------------------------------------------------------------------------------------

+ vectors Timers (4)

        INDEX  TIMER_LIST_ADDR    EXPIRES        EXPIRES(s)     FUNCTION                                  WORK
        147    ffffacc1ee663c10   4294902956     42.644s        delayed_work_timer_fn                     crng_reseed
        165    ffff700dff082498   4294900013     30.872s        delayed_work_timer_fn                     kfree_rcu_monitor
        167    ffff700c80d74a08   4294900131     31.344s        delayed_work_timer_fn                     wb_workfn
        184    ffff700c9df838a0   4294901235     35.76s         <dynamic module>

------------------------------------------------------------------------------------------------------------------------
CPU 1(tvec_base: ffff700dff0a15c0 timer_jiffies: 4294900025(30.92s) next_timer: 4294900152(31.428s) active_timers: NA)
------------------------------------------------------------------------------------------------------------------------

+ vectors Timers (2)

        INDEX  TIMER_LIST_ADDR    EXPIRES        EXPIRES(s)     FUNCTION                                  WORK
        65     ffff700c82ac2898   4294900226     31.724s        <dynamic module>
        119    ffff80008072bd68   4294900149     31.416s        process_timeout

------------------------------------------------------------------------------------------------------------------------
CPU 2(tvec_base: ffff700dff0c05c0 timer_jiffies: 4294900063(31.072s) next_timer: 4294900065(31.08s) active_timers: NA)
------------------------------------------------------------------------------------------------------------------------

+ vectors Timers (3)

        INDEX  TIMER_LIST_ADDR    EXPIRES        EXPIRES(s)     FUNCTION                                  WORK
        33     ffff8000800d3cd8   4294900064     31.076s        process_timeout
        167    ffff700dff0c0498   4294900153     31.432s        delayed_work_timer_fn                     kfree_rcu_monitor
        182    ffff800080753d08   4294901060     35.06s         process_timeout

------------------------------------------------------------------------------------------------------------------------
CPU 3(tvec_base: ffff700dff0df5c0 timer_jiffies: 4294900001(30.824s) next_timer: 4294902528(40.932s) active_timers: NA)
------------------------------------------------------------------------------------------------------------------------

+ vectors Timers (3)

        INDEX  TIMER_LIST_ADDR    EXPIRES        EXPIRES(s)     FUNCTION                                  WORK
        140    ffff8000849fbb78   4294902482     40.748s        process_timeout
        148    ffff800080bbbd48   4294903020     42.9s          process_timeout
        239    ffffacc1ee771720   4294925741     133.784s       delayed_work_timer_fn                     check_lifetime
```

Figure : Timer list of cores (timerlist.txt)

```
CPU 0
pool 0
BUSY Workqueue worker: kworker/0:0 current_work: deferred_probe_timeout_work_func last_func: ('kernfs_notify_workfn', 0)
IDLE Workqueue worker: kworker/0:2 current_work: (None) last_func: ('kernfs_notify_workfn', 0)
IDLE Workqueue worker: kworker/0:1 current_work: (None) last_func: ('sync_rcu_do_polled_gp', 0)
       Pending entry: v.v (struct work_struct)0xffffacc1ee4a8570 kernfs_notify_workfn  Line 376 of \"/usr/src/kernel/include/linux/spinlock.h\"
        Pending entry: v.v (struct work_struct)0xffff700c9ba964b8 qcom_glink_rx_done_work  Line 524 of \"/usr/src/kernel/drivers/rpmsg/qcom_glink_native.c\"
        Pending entry: v.v (struct work_struct)0xffff700c836f3a98 qcom_glink_work  Line 1667 of \"/usr/src/kernel/drivers/rpmsg/qcom_glink_native.c\"
 pool 1
BUSY Workqueue worker: kworker/0:0 current_work: deferred_probe_timeout_work_func last_func: ('kernfs_notify_workfn', 0)
IDLE Workqueue worker: kworker/0:1H current_work: (None) last_func: ('blk_mq_requeue_work', 0)
IDLE Workqueue worker: kworker/0:0H current_work: (None) last_func: ('fill_page_cache_func', 0)

CPU 1
pool 0
IDLE Workqueue worker: kworker/1:8 current_work: (None) last_func: ('vmstat_update', 0)
IDLE Workqueue worker: kworker/1:7 current_work: (None) last_func: ('css_free_rwork_fn', 0)
IDLE Workqueue worker: kworker/1:9 current_work: (None) last_func: None
IDLE Workqueue worker: kworker/1:2 current_work: (None) last_func: ('css_release_work_fn', 0)
IDLE Workqueue worker: kworker/1:6 current_work: (None) last_func: ('cgroup_bpf_release', 0)
IDLE Workqueue worker: kworker/1:5 current_work: (None) last_func: ('cgroup_bpf_release', 0)
IDLE Workqueue worker: kworker/1:4 current_work: (None) last_func: ('cgroup_bpf_release', 0)
IDLE Workqueue worker: kworker/1:1 current_work: (None) last_func: ('cgroup_bpf_release', 0)
IDLE Workqueue worker: kworker/1:3 current_work: (None) last_func: ('css_killed_work_fn', 0)
IDLE Workqueue worker: kworker/1:0 current_work: (None) last_func: ('cgroup_bpf_release', 0)
pool 1
IDLE Workqueue worker: kworker/1:1H current_work: (None) last_func: ('blk_mq_timeout_work', 0)
IDLE Workqueue worker: kworker/1:0H current_work: (None) last_func: ('blk_mq_run_work_fn', 0)

CPU 2
pool 0
IDLE Workqueue worker: kworker/2:2 current_work: (None) last_func: ('vmstat_update', 0)
IDLE Workqueue worker: kworker/2:0 current_work: (None) last_func: ('release_one_tty', 0)
IDLE Workqueue worker: kworker/2:3 current_work: (None) last_func: ('kernfs_notify_workfn', 0)
IDLE Workqueue worker: kworker/2:1 current_work: (None) last_func: ('device_link_release_fn', 0)
IDLE Workqueue worker: kworker/2:4 current_work: (None) last_func: None
pool 1
IDLE Workqueue worker: kworker/2:1H current_work: (None) last_func: ('blk_mq_requeue_work', 0)
IDLE Workqueue worker: kworker/2:0H current_work: (None) last_func: ('blk_mq_run_work_fn', 0)

CPU 3
pool 0
IDLE Workqueue worker: kworker/3:7 current_work: (None) last_func: ('vmstat_update', 0)
IDLE Workqueue worker: kworker/3:6 current_work: (None) last_func: ('kernfs_notify_workfn', 0)
IDLE Workqueue worker: kworker/3:3 current_work: (None) last_func: ('cgroup_bpf_release', 0)
IDLE Workqueue worker: kworker/3:2 current_work: (None) last_func: ('css_release_work_fn', 0)
IDLE Workqueue worker: kworker/3:4 current_work: (None) last_func: ('cgroup_bpf_release', 0)
IDLE Workqueue worker: kworker/3:5 current_work: (None) last_func: ('cgroup_bpf_release', 0)
IDLE Workqueue worker: kworker/3:0 current_work: (None) last_func: ('cgroup_bpf_release', 0)
```

Figure : Workqueue (dmesg_TZ.txt)

*mem_stat - Notepad

File Edit Format View Help

```
Total RAM                    :      6423 MB
Free memory:                 :      5427 MB

Total Slab memory:           :       125 MB
Total DMA memory             :        30 MB
KGSL                         :         0 MB
ZRAM compressed              :         0 MB
vmalloc                      :        39 MB
Others                       :       639 MB
Cached                       :       532 MB

Total Unaccounted Memory     :       163 MB
```

Figure : Memory statistics (mem_stat.txt)

```
----------begin Schedinfo----------

CPU Frequency information:
----------
CPU:0   Governor:schedutil       cur_freq:1958400, max_freq:1958400, min_freq300000  cpuinfo: min_freq:300000, max_freq:1958400
        Capacity: capacity_orig:382, cur_cap:379, arch_scale:382

CPU:1   Governor:schedutil       cur_freq:1958400, max_freq:1958400, min_freq300000  cpuinfo: min_freq:300000, max_freq:1958400
        Capacity: capacity_orig:382, cur_cap:382, arch_scale:382

CPU:2   Governor:schedutil       cur_freq:1958400, max_freq:1958400, min_freq300000  cpuinfo: min_freq:300000, max_freq:1958400
        Capacity: capacity_orig:382, cur_cap:382, arch_scale:382

CPU:3   Governor:schedutil       cur_freq:1958400, max_freq:1958400, min_freq300000  cpuinfo: min_freq:300000, max_freq:1958400
        Capacity: capacity_orig:382, cur_cap:382, arch_scale:382

CPU:4   Governor:schedutil       cur_freq:691200, max_freq:2400000, min_freq691200  cpuinfo: min_freq:691200, max_freq:2400000
        Capacity: capacity_orig:890, cur_cap:888, arch_scale:890

CPU:5   Governor:schedutil       cur_freq:691200, max_freq:2400000, min_freq691200  cpuinfo: min_freq:691200, max_freq:2400000
        Capacity: capacity_orig:890, cur_cap:890, arch_scale:890

CPU:6   Governor:schedutil       cur_freq:691200, max_freq:2400000, min_freq691200  cpuinfo: min_freq:691200, max_freq:2400000
        Capacity: capacity_orig:890, cur_cap:890, arch_scale:890

CPU:7   Governor:schedutil       cur_freq:2707200, max_freq:2707200, min_freq806400  cpuinfo: min_freq:806400, max_freq:2707200
        Capacity: capacity_orig:1024, cur_cap:1017, arch_scale:1024

------------end Schedinfo----------
```

Figure : CPU frequency (dmesg_TZ.txt)

```
Launch_t32.bat
start C:\T32\bin\windows64\t32MARM64.exe -c RAM_DUMP_FOLDER/t32_config.t32, RAM_DUMP_FOLDER/t32_startup_script.cmm

==============================================================
t32_config.t32
==============================================================
cd c:\RAM_DUMP_FOLDER
title "c:\RAM_DUMP_FOLDER"
sys.cpu CORTEXA53
SYStem.Option MMUSPACES ON
SYStem.Option ZONESPACES OFF
sys.up
data.load.binary c:\RAM_DUMP_FOLDER\OCIMEM.BIN 0x14680000
data.load.binary c:\RAM_DUMP_FOLDER\PIMEM.BIN 0x1c000000
data.load.binary c:\RAM_DUMP_FOLDER\DDRCS0_0.BIN 0x80000000
data.load.binary c:\RAM_DUMP_FOLDER\DDRCS0_1.BIN 0x100000000
data.load.binary c:\RAM_DUMP_FOLDER\DDRCS1_0.BIN 0x180000000
data.load.binary c:\RAM_DUMP_FOLDER\DDRCS1_1.BIN 0x200000000

==============================================================
t32_startup_script.cmm
==============================================================
OS=
ID=T32_1000002
TMP=C:\TEMP
SYS=C:\T32
HELP=C:\T32\pdf
PBI=SIM
SCREEN=
FONT=LARGE
HEADER=Trace32-ScorpionSimulator
PRINTER=WINDOWS
RCL=NETASSIST
PACKLEN=1024
PORT=26288
```

Figure : TRACE32 simulator launcher (launch_t32.bat)

3.9 Parse RAM dumps using QCAP

QCAP is a tool to parse logs from all subsystems and determine on which subsystem the crash occurred first.

This tool is available to licensed developers with authorized access. For more information about how to parse subsystem dumps using QCAP, see Qualcomm Linux Debug Guide - Addendum.

3.10 Crash utility

Crash utility is an open-source tool to debug kernel using a gdb-based command-line interface over RAM dumps.

Prerequisites

- `kaslr_offset` and `kimage_voffset` values

  By default, the Qualcomm Linux build has the Kernel Address Space Layout Randomization (KASLR) feature enabled. For the crash utility to work on a KASLR-enabled kernel, you need the values of the `kaslr_offset` and `kimage_voffset` parameters, which you can extract from the `dmesg_TZ.txt` RAMparser output file.

  The following is an excerpt from the sample RAMparser output file, `dmesg_TZ.txt`, providing the `kaslr_offset` and `kimage_voffset` values.

  ```
  The kaslr_offset extracted is: 0x340c3d320000
  ...
  The kimage_voffset extracted is: ffffb40bbf600000
  ```

- Base address of the kernel binaries (`DDRCS*.BIN`)

  When you use PCAT to capture the RAM dump, it also captures the `dump_info.txt` and `load.cmm` files. These files contain the base address of the kernel binaries (`DDRCS*.BIN`).

Download and build crash utility on Linux host

To download and build the crash utility, run the following commands:

```
git  clone https://github.com/crash-utility/crash
```

```
make target=ARM64
```

```
make extensions=ARM64
```

For more information, see crash/README at master · crash-utility/crash· GitHub.

Run crash utility

The following is the sample command to run the crash utility on an Ubuntu host:

```
./crash <PATH>/vmlinux <PATH>/DDRCS0_0.BIN@0x80000000,<PATH>/DDRCS1_
0.BIN@0x100000000,<PATH>/DDRCS1_1.BIN@0x180000000 -m vabits_actual=48
-m max_physmem_bits=48 -m kimage_voffset=0xffffb40bbf600000  --
kaslr=0x340c3d320000
```

- Replace `<PATH>` with the vmlinux RAM dump path.
- Extract `kimage_voffset`, and `kaslr` from the `dmesg_TZ.txt` RAMparser output file.
- Extract the DDR offsets such as `0x80000000` from the `dump_info.txt` file available in the RAM dump collected using PCAT.

Sample output:

```
crash 8.0.4
Copyright (C) 2002-2022 Red Hat, Inc.
Copyright (C) 2004, 2005, 2006, 2010 IBM Corporation
Copyright (C) 1999-2006 Hewlett-Packard Co
Copyright (C) 2005, 2006, 2011, 2012 Fujitsu Limited
Copyright (C) 2006, 2007 VA Linux Systems Japan K.K.
Copyright (C) 2005, 2011, 2020-2022 NEC Corporation
Copyright (C) 1999, 2002, 2007 Silicon Graphics, Inc.
Copyright (C) 1999, 2000, 2001, 2002 Mission Critical Linux, Inc.
Copyright (C) 2015, 2021 VMware, Inc.
This program is free software, covered by the GNU General Public License,
and you are welcome to change it and/or distribute copies of it under
certain conditions. Enter "help copying" to see the conditions.
This program has absolutely no warranty. Enter "help warranty" for details.

NOTE: setting vabits_actual to: 48

NOTE: setting max_physmem_bits to: 48
GNU gdb (GDB) 10.2
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=x86_64-pc-linux-gnu --target=aarch64-elf-linux".
Type "show configuration" for configuration details.
Find the GDB manual and other documentation resources online at:
        <http://www.gnu.org/software/gdb/documentation/>.
```

```
For help, type "help".
Type "apropos word" to search for commands related to "word"...

WARNING: cpu 0: cannot find NT_PRSTATUS note
WARNING: cpu 1: cannot find NT_PRSTATUS note
WARNING: cpu 2: cannot find NT_PRSTATUS note
WARNING: cpu 3: cannot find NT_PRSTATUS note
WARNING: cpu 4: cannot find NT_PRSTATUS note
WARNING: cpu 5: cannot find NT_PRSTATUS note
WARNING: cpu 6: cannot find NT_PRSTATUS note
WARNING: cpu 7: cannot find NT_PRSTATUS note
          KERNEL: /test/vmlinux [TAINTED]
     DUMPFILES: /var/tmp/ramdump_elf_PBaLfj [temporary ELF header]
                /test/DDRCS0_0.BIN
                /test/DDRCS1_0.BIN
                /test/DDRCS1_1.BIN
          CPUS: 8 [OFFLINE: 7]
          DATE: Sun Jan 6 05:30:34 +0530 1980
          UPTIME: 00:00:48
LOAD AVERAGE: 1.60, 0.51, 0.18
          TASKS: 532
     RELEASE: 6.6.17-debug
     VERSION: #1 SMP PREEMPT Mon Mar 25 04:52:52 UTC 2024
     MACHINE: aarch64 (unknown Mhz)
          MEMORY: 5.5 GB
          PANIC: "Kernel panic - not syncing: sysrq triggered crash"
               PID: 1419
     COMMAND: "sh"
          TASK: ffff1756f61ea000 [THREAD_INFO: ffff1756f61ea000]
               CPU: 0
          STATE: TASK_RUNNING (PANIC)
```

```
crash> man
```

```
* files mod sbitmapq union
alias foreach mount search vm
ascii fuser net set vtop
bpf gdb p sig waitq
bt help ps struct whatis
btop ipcs pte swap wr
dev irq ptob sym q
dis kmem ptov sys
eval list rd task
exit log repeat timer
extend mach runq tree

crash version: 8.0.4 gdb version: 10.2
For help on any command above, enter "help <command>".
For help on input options, enter "help input".
For help on output options, enter "help output".
```

```
crash> log
```

```
[ 0.000000] Booting Linux on physical CPU 0x0000000000 [0x412fd050]
[ 0.000000] Linux version 6.6.17-debug (oe-user@oe-host) (aarch64-qcom-linux-gcc (GCC) 11.4.0, GNU ld (GNU Binutils) 2.38.20220708) #1 SMP PREEMPT Mon Mar 25 04:52:52 UTC 2024
[ 0.000000] KASLR enabled
[ 0.000000] Machine model: Qualcomm Technologies, Inc. Robotics RB3gen2 addons vision mezz platform
[ 0.000000] efi: EFI v2.7 by Qualcomm Technologies, Inc.
[ 0.000000] efi: MEMATTR=0x9ccf6018 INITRD=0x9cceaf18 RNG=0x9cce3018 MEMRESERVE=0x9ccea818
[ 0.000000] random: crng init done
[ 0.000000] Reserved memory: created CMA memory pool at 0x00000000fd000000, size 12 MiB
[ 0.000000] OF: reserved mem: initialized node adsp-heap, compatible id shared-dma-pool
```

```
crash> p memdump
```

```
memdump = $1 = {
table_phys = 4110417920,
table = 0xffff80008087d000
}
```

For more information about the crash utility, see the following:

- https://crash-utility.github.io/
- https://man7.org/linux/man-pages/man8/crash.8.html

3.11 Subsystem dumps

This feature is available to licensed developers with authorized access. For more information about how to enable and capture coredumps of subsystems, see Qualcomm Linux Debug Guide - Addendum.

3.12 Debug using OpenOCD

OpenOCD is an open-source on-chip debugger that provides debugging, in-system programming, and boundary-scan testing for embedded devices. OpenOCD supports a range of interfaces, such as JTAG and SWD, which makes it a versatile debugging tool. OpenOCD supports GDB, LLDB and other debugging tools, which provides the developers with a familiar and powerful debugging environment. For more information, see OpenOCD.

Review the following information before using the OpenOCD.

- Use Ubuntu Linux host computer for the debug setup.
- Qualcomm supports OpenOCD version 0.12.0. For more information, see `openocd/configure.ac`.
- OpenOCD is supported using QC Embedded USB debugger (EUD) solution over USB Type-C, LLDB, and GDB as command line debuggers.
- Keep the Qualcomm Linux build ready. For more information, see Qualcomm Linux Build Guide.

Known limitations

- Qualcomm supports OpenOCD debug for APSS Linux kernelspace. Kernel modules (`.ko`) debug isn't supported in this release.
- At present, it supports only QCS6490/QCM6490 chipset-based target, such as Qualcomm® RB3 Gen 2.
- Kernel debug during bootup isn't supported until you manually enable EUD in the console shell or adb.
- When you enable EUD, the system disables the USB host mode and adb.

Setup Ubuntu host

Install the following pre-requisite package:

```
sudo apt-get install libusb-1.0

sudo apt-get install libftdi-dev

sudo apt-get install libftdi1

sudo apt-get install pkg-config
```

Ensure that the version of `autoconf` is at least v2.71.

```
autoconf --version
```

Download OpenOCD

The following are the commands to download OpenOCD that supports Qualcomm chipsets:

```
git clone https://git.codelinaro.org/clo/la/openocd-org/openocd.git -
b qcom_changes

cd openocd

git checkout 3124da65ca8bfa297624991904d5fb906f0161af

git submodule update --init --recursive
```

Build OpenOCD

After downloading the source code, do the following to compile the OpenOCD:

1. Go to the OpenOCD directory.

```
cd openocd
```

2. Generate the configuration scripts.

```
./bootstrap
```

3. Configure the build. This step generates the makefile required to build OpenOCD, with the options you provide. In the following command, the system incorporates the EUD for debugging.

```
./configure --disable-werror --enable-eud --disable-
internal-libjaylink --disable-jlink --disable-ftdi --
disable-dummy --disable-rshim --disable-stlink --disable-
ti-icdi --disable-ulink --disable-usb-blaster-2 --
disable-ft232r --disable-vsllink --disable-xds110 --
disable-cmsis-dap-v2 --disable-osbdm --disable-opendous -
-disable-aice --disable-usbprog --disable-rlink --
disable-armjtagew --disable-cmsis-dap --disable-nulink --
disable-kitprog --disable-usb-blaster --disable-presto --
disable-openjtag --disable-parport --disable-parport-
giveio --disable-jtag_vpi --disable-jtag_dpi --disable-
amtjtagaccel --disable-zy1000-master --disable-zy1000 --
disable-ioutil --disable-bcm2835gpio --disable-imx_gpio -
-disable-ep93xx --disable-at91rm9200 --disable-gw16012 --
disable-oocd_trace --disable-buspirate --disable-
sysfsgpio --disable-xlnx-pcie-xvc --disable-minidriver-
dummy --disable-remote-bitbang --disable-parport-ppdev --
disable-esp-usb-jtag
```

Note: Disable the adapters that aren't required in Qualcomm Linux.

For more information about the build configuration, run the `./configure -help` command.

4. Compile the source code.

```
make
```

5. Clean the build.

```
make clean
```

When the build succeeds, it generates the OpenOCD binary in the `openocd/src/` directory.

Setup command line debuggers

Qualcomm supports LLDB, and GDB debuggers on the host computer. For more information about the setup, see the following:

- LLDB

  For installation, go to LLVM packages. To autoinstall script for the version-19, see the following sample example:

```
/# wget https://apt.llvm.org/llvm.sh
/# chmod +x llvm.sh
/# sudo ./llvm.sh 19
/# export PATH=/usr/lib/llvm-19/bin:$PATH
/# lldb -version
lldb version 19.1.7
```

- GDB

  For installation, go to GNU toolchain downloads. To install the version-14.2, see the following sample example:

```
/# wget https://developer.arm.com/-/media/Files/
downloads/gnu/14.2.rel1/binrel/arm-gnu-toolchain-14.2.
rel1-x86_64-aarch64-none-elf.tar.xz
/# tar -xf arm-gnu-toolchain-14.2.rel1-x86_64-aarch64-
none-elf.tar.xz
```

The following are the optional commands:

```
/# sudo mv arm-gnu-toolchain-14.2.rel1-x86_64-aarch64-none-elf /opt/

/# export PATH=/opt/arm-gnu-toolchain-14.2.rel1-x86_64-aarch64-none-
elf/bin:$PATH

/# aarch64-none-elf-gdb -version
```

Sample output:

GNU gdb (Arm GNU Toolchain **14.2.Rel1** (Build arm-14.52)) 15.2.90.20241130-git

Setup APPS kernel build

Qualcomm recommends to use a debug build using the `export DEBUG_BUILD=1` command. For more information, see Qualcomm Linux Build Guide and Qualcomm Linux Kernel Guide.

To debug using OpenOCD, add the following parameters in the kernel command line. This ensures that only a single core boots up without KASLR.

```
nokaslr nosmp nr_cpus=1 maxcpus=1
```

Add the `nokaslr nosmp nr_cpus=1 maxcpus=1` command line parameters at the target specific file at `layers/meta-qcom-hwe/conf/machine/include`.

Use the `DBG_CMDLINE` or `KERNEL_CMDLINE_EXTRA` macro present in the file.

The following is the sample example of the `qcom-qcs6490.inc` machine.

```
DBG_CMDLINE = ${@oe.utils.conditional('DEBUG_BUILD',
'1','earlycon page_owner=on module.sig_enforce=0
nokaslr nosmp nr_cpus=1 maxcpus=1 qcom_scm.download_
mode=1 slub_debug=FZP,zs_handle,zspage;FZPU','',d)}
```

To enable EUD, see the following:

Enable EUD on host computer

The host computer doesn't require any extra drivers to detect the EUD target device.

On-target device

By default, EUD isn't enabled in the Linux kernel. To enable the EUD support, add the following patches manually in the kernel devicetree. Apply the following three patches, then rebuild, and reflash the device.

1. Go to the `arch/arm64/boot/dts/qcom/`

`qcs6490-addons-rb3gen2.dtsi` file and see the changes at `+/-`.

```
cooling-maps {

  map0 {

       trip = <&b_bcl_lvl2>;

      cooling-device = <&cdsp_sw 5 5>;

       };

    };

   };

};

+  &eud {

+      status = "okay";

+  };
```

1. Go to the `arch/arm64/boot/dts/qcom/qcs6490-rb3gen2.dts` file and see the changes at `+/-`.

```
 &usb_1 {

    reg = <0x0 0x0a600000 0x0
0x200000>;

    interrupts-extended = <&intc
GIC_SPI 133 IRQ_TYPE_LEVEL_HIGH>,

        <&intc GIC_SPI 131 IRQ_TYPE_
LEVEL_HIGH>,

        <&pdc 14 IRQ_TYPE_LEVEL_
HIGH>,

        <&pdc 15 IRQ_TYPE_EDGE_BOTH>
,
```

```
        <&pdc 17 IRQ_TYPE_EDGE_BOTH>
;

    interrupt-names = "dwc_usb3",

        "hs_phy_irq",

        "dp_hs_phy_irq",

        "dm_hs_phy_irq",

        "ss_phy_irq";

    iommus = <&apps_smmu 0xe0 0x0>;

     snps,dis_u2_susphy_quirk;

     snps,dis_enblslpm_quirk;

     phys = <&usb_1_hsphy>, <&usb_1_
qmpphy QMP_USB43DP_USB3_PHY>;

     phy-names = "usb2-phy", "usb3-
phy";

     maximum-speed = "super-speed";

     snps,dis-u2-entry-quirk;

     snps,dis-u1-entry-quirk;

     dr_mode = "otg";

     usb-role-switch;

     qcom,enable-rt;

     wakeup-source;

 +        role-switch;

     status = "okay";
```

```
     ports {

           #address-cells = <1>;

           #size-cells = <0>;

           port@0 {

                 reg = <0>;

                 usb_1_dwc3_hs:
endpoint {

                 };

 +                port@1 {

 +                        usb1_role_
switch: endpoint {

 +                        };

 +                 };

    };

 };

 ...

 &usb_1_dwc3_hs {

     remote-endpoint = <&pmic_glink_
hs_in>;

 };

+ &usb1_role_switch {

+         remote-endpoint = <&eud_
ep>;

+ };
```

```
...

    port@0 {
            reg = <0>;
            usb2_port: endpoint  {
                  remote-endpoint =
<&usb2_port_connector>;
            };
    };
    port@1  {
            usb2_role_switch:
endpoint  {
 -             remote-endpoint = <&
eud_ep>;
             };
    };
```

2. Go to the `/arch/arm64/boot/dts/qcom/sc7280.dtsi` file and see the changes at `+/-`.

```
eud: eud@88e0000 {

-                   compatible =
"qcom,eud";

+                   compatible =
"qcom,secure-eud";

        reg = <0 0x88e0000 0 0x2000>
,

              <0 0x88e2000 0 0x1000>
;

         interrupts-extended = <&pdc
11 IRQ_TYPE_LEVEL_HIGH>;

         status = "disabled";

         ports {

                #address-cells = <1>;

                #size-cells = <0>;
```

```
                port@0 {

                        reg = <0>;

                       eud_ep:
endpoint {

          -
                        remote-
endpoint = <&usb2_role_switch>;

          +
                        remote-
endpoint = <&usb1_role_switch>;

                       };

...

         usb_1_dwc3: usb@a600000 {

                compatible = "snps,
dwc3";

                reg = <0 0x0a600000 0
0xe000>;

                interrupts = <GIC_SPI
133 IRQ_TYPE_LEVEL_HIGH>;

                iommus = <&apps_smmu
0xe0 0x0>;

                snps,dis_u2_susphy_
quirk;

                snps,dis_enblslpm_
quirk;

                snps,parkmode-
disable-ss-quirk;

                phys = <&usb_1_hsphy>
```

```
, <&usb_1_qmpphy

QMP_USB43DP_USB3_PHY>;

               phy-names = "usb2-phy
", "usb3-phy";

               maximum-speed =
"super-speed";

+                          role-
switch;

           ports {

               #address-cells = <1>;

               #size-cells = <0>;

               port@0 {

                      reg = <0>;

                       usb_1_dwc3_
hs: endpoint {

                      };

                  };

+
port@1 {

+
      usb1_role_switch: endpoint {

+
      };

+                                  }
;

         };
```

```
        };

    };
```

Enable EUD on the device using console shell or adb.

```
/# echo 1 >/sys/bus/platform/devices/
88e0000.eud/enable
```

Note:

- With the mentioned three patchsets, the role switch has a limitation to send notification for both the glink and extcon. After enabling EUD, USB host mode won't work because it doesn't receive the role from the glink on the port connector.
- With every system restart, it's required to enable the EUD using these adb commands.

Setup target device

For debugging applications, it's required to remove modem binaries from the target device. Use the following sample commands to remove modem binaries before debugging the OpenOCD.

Sign in using UART:

```
/# mount -o remount,rw /
/# cd /lib/firmware/qcom/qcm6490/ && rm -rf modem*
/# cd /lib/firmware/qcom/qcm6490/ && ls
/# cd /lib/firmware/qcom/qcs6490/ && rm -rf modem*
/# cd /lib/firmware/qcom/qcs6490/ && ls
/# reboot
```

Debug steps

Once you complete the setup, debug the Linux kernel. The following are the sample debug commands for QCS6490 (Qualcomm RB3 Gen 2) target device:

1. Flash the device with the implemented APPS kernel setup modifications. For more information, see Setup APPS kernel build.
2. Attach the Qualcomm RB3 Gen 2 device with the host computer using the USB Type-C cable.
3. Enable the EUD on target by using console shell or adb:

   ```
   /# echo 1 >/sys/bus/platform/devices/88e0000.eud/enable
   ```

   The following is the expected output on the host computer, when you enable the 9501 EUD port:

   ```
   /#lsusb
   Bus 001 Device 038: ID 05c6:9501 Qualcomm, Inc.
   Bus 001 Device 037: ID 05c6:9500 Qualcomm, Inc.
   ```

4. Run the OpenOCD on host computer.

   ```
   cd openocd/src
   ```

   For EUD:

   ```
   sudo ./openocd -f ../tcl/interface/eud.cfg -f ../tcl/target/
   qualcomm/qcs6490.cfg
   ```

   Sample output:

```
Open On-Chip Debugger 0.12.0-01020-g3124da65c (2025-03-21-15:28)
Licensed under GNU GPL v2
For bug reports, read
http://openocd.org/doc/doxygen/bugs.html
Info : only one transport option; autoselect 'swd'
Warn : Transport "swd" was already selected
force hard breakpoints
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : Using EUD 2.1.7
Error: Translation from adapter speed to khz not implemented
Info : adapter-specific clock speed value 6
Info : SWD DPIDR 0x5ba02477
Info : QCS6490.cpu0: hardware has 6 breakpoints, 4 watchpoints
Info : starting gdb server for QCS6490.cpu0 on 3333
Info : Listening on port 3333 for gdb connections
Info : QCS6490.cpu0 cluster 0 core 0 multi core
QCS6490.cpu0 halted in AArch64 state due to debug-request, current mode: EL0T
cpsr: 0x60001000 pc: 0xffffffc080010c00
MMU: enabled, D-Cache: enabled, I-Cache: enabled
```

Note: In case of a connection failure with OpenOCD, try rerunning the OpenOCD command.

5. Two debuggers LLDB, and GDB support the host computer. For more information, see the following:
   - Using LLDB
     a. Open new terminal and run the LLDB with the following command.

        ```
        /# lldb
        ```

     b. To load the application symbols, run the following command.

        ```
        (lldb) target create <vmlinux_path>
        ```

     c. Map the source code using the following command.

        ```
        (lldb) settings set target.source-map <build_path_
        from_vmlinux> <local_path_to_source_code>
        ```

In this command, `<build_path_from_vmlinux>` is `/usr/src/kernel/<SUFFIX>`, where `SUFFIX` is the actual code directory in the kernel. For example, `/usr/src/kernel/drivers/usb/gadget`.

d. Connect to the OpenOCD.

```
(lldb) gdb-remote 3333
```

e. For more LLDB commands, see the Tutorial.

- Using GDB

a. Open new terminal and run the GDB with the following command.

```
/# aarch64_none_elf_gdb
```

b. To load application symbols, run the following command.

```
(gdb) add-symbol-file <vmlinux_path>
```

c. Map the source code using the following command.

```
(gdb) set substitute-path <build_path_from_vmlinux>
<local_path_to_source_code>
```

In this command, `<build_path_from_vmlinux>` is `/usr/src/kernel/<SUFFIX>`, where `SUFFIX` is the actual code directory in the kernel. For example `/usr/src/kernel/drivers/usb/gadget`.

d. Connect to OpenOCD.

```
(gdb) target remote localhost:3333
```

e. For more GDB commands, see the GDB command reference.

4 Debug common system issues

Some common system issues are watchdog timeout, bus hang, timeout error, and hardware reset. The following sections provide information about how to identify and debug such system issues.

4.1 Watchdog issues

A watchdog (WD) is a fixed-length counter that allows a system to recover from an unexpected hardware or software catastrophe. Unless the system periodically pets the watchdog timer, it assumes a catastrophe and resets the subsystem or the entire system, depending on the triggered watchdog.

The following are the different types of watchdog implementations:

- Hardware watchdog
- Software watchdog
- Bark, and
- Bite

The following table summarizes different types of watchdog implementation.

Table : Watchdog implementations

Types of watchdogs	Timeout duration (in seconds)	Owner	When expired	Result
Nonsecure WD bark	11	HLOS	IRQ to Qualcomm TEE	HLOS falls to Panic
Nonsecure WD bite	12	HLOS	Fast interrupt request (FIQ) to Qualcomm TEE	Qualcomm TEE asserts PS_HOLD
Secure WD bark	6	Qualcomm TEE	FIQ to Qualcomm TEE	Qualcomm TEE just pets secure WD
Secure WD bite	22	Qualcomm TEE	Asserting PS_HOLD	PMIC resets the system

The system uses both hardware and software watchdogs. For example, modem DSP (mDSP) implements both software and hardware watchdogs. The hardware watchdog module ensures that

the processor is active and consists of a timer that counts down from a predetermined value. If the corresponding CPU core doesn't reset the timer, it eventually counts to 0 (zero) and triggers a watchdog timeout.

Watchdog for application processor CPU

Nonsecure hardware watchdog

- Every 10 seconds, the HLOS triggers a timer event to pet the nonsecure hardware watchdog. If the HLOS doesn't pet the nonsecure watchdog for 11 seconds, the nonsecure watchdog bark fires and the HLOS must handle it. If the HLOS can't handle it, the HLOS falls into panic.
- If the HLOS is unable to handle nonsecure watchdog bark, it triggers a nonsecure watchdog bite and sends it to Qualcomm TEE, causing the Qualcomm TEE to fall into a fatal error.
- You can customize the watchdog pet and bark time using the kernel configuration options. For example, the following configuration sets the bark time to 13 seconds and the pet time to 11 seconds:

```
CONFIG_QCOM_WATCHDOG_BARK_TIME=13000
```

```
CONFIG_QCOM_WATCHDOG_PET_TIME=11000
```

Secure hardware watchdog

- Every 6 seconds, Qualcomm TEE triggers a secure watchdog bark as a fast interrupt request (FIQ). The FIQ handler in the Qualcomm TEE pets the secure hardware Watchdog. This issue isn't an error or fatal issue.
- If Qualcomm TEE can't handle the secure watchdog bark for 22 seconds, the secure watchdog bite expires. Then, the PMIC asserts the PS_HOLD pin, and eventually, the entire system is reset.

The complete functionality of this feature is available to licensed developers with authorized access. For more information about debugging watchdog issues, see Qualcomm Linux Debug Guide - Addendum.

4.2 Bus hang and timeout error

The SNoC, CNoC, xPU, TBU, and AHB are the system infrastructure components on the device, which are responsible for operations such as:

- Bus transaction
- Address translation
- Memory protection

Some failures or timeout on these components may cause system errors, which the system reports to the Qualcomm TEE.

This feature is available to licensed developers with authorized access. For more information about debugging bus hang and timeout errors, see Qualcomm Linux Debug Guide - Addendum.

4.3 Hardware reset

A secure watchdog, temperature sensor (TSENS), or PMIC issues can cause a hardware reset. The debugging approach for hardware reset issues depends on the cause of the hardware reset. Therefore, identifying the cause of the hardware reset is crucial.

This feature is available to licensed developers with authorized access. For more information about debugging hardware reset issues, see Qualcomm Linux Debug Guide - Addendum.

5 Debug non-HLOS

To debug non-HLOS issues, see the corresponding documentation as listed in the following table.

Table : Documents to debug issues in non-HLOS

Non-HLOS software	Document
aDSP	Qualcomm Linux Debug Guide - Addendum
AOP	Qualcomm Linux Debug Guide - Addendum
Qualcomm® Bluetooth	Qualcomm Linux Bluetooth Guide
DDR	Qualcomm Linux Memory Guide
Diag	Qualcomm Linux Debug Guide - Addendum
WLAN	Qualcomm Linux Wi-Fi Guide
XBL	Qualcomm Linux Boot Guide

6 References

6.1 Related documents

Title	Number
Qualcomm Technologies, Inc.	
Qualcomm Linux Kernel Guide	80-70018-3
Qualcomm Linux Boot Guide	80-70018-4
Qualcomm Linux Security Guide	80-70018-11
Qualcomm Linux Bluetooth Guide	80-70018-13
Qualcomm Linux Wi-Fi Guide	80-70018-14
Qualcomm Linux Yocto Guide	80-70018-27
Resources	
Qualcomm® Package Manager 3 Documentation	
Arm System Memory Management Unit Architecture Specification, IHI0062D	
https://docs.kernel.org/trace/coresight/coresight-perf.html	
https://docs.kernel.org/trace/coresight/index.html	
https://perf.wiki.kernel.org/index.php/Tutorial	
https://man7.org/linux/man-pages/man1/ltrace.1.html	
https://man7.org/linux/man-pages/man1/gdb.1.html	
https://man7.org/linux/man-pages/man5/core.5.html	
https://man7.org/linux/man-pages/man1/gdbserver.1.html	
https://www.kernel.org/doc/Documentation/ABI/testing/sysfs-devices-system-cpu	

Title	Number
https://man7.org/linux/man-pages/man8/crash.8.html	

6.2 Acronyms and terms

Acronym or term	Definition
aDSP	Application digital signal processor
AHB	Advanced High-performance Bus
AOP	Always-on-processor
APSS	Application processor subsystem
cDSP	Compute DSP
CMA	Contiguous memory allocator
CNoC	Config network-on-chip
DCC	Data capture and compare
DDR	Double data rate
EUD	Embedded USB debugger
FAR	Fault address register
FIQ	Fast interrupt request
FSR	Fault status register
GDB	GNU debugger
GFP	Get free pages
HLOS	High-level OS
IRQ	Interrupt request
KASAN	Kernel address sanitizer
KASLR	Kernel address space layout randomization
LPASS	Low-power audio subsystem
NoC	Network-on-chip: It's a bus connecting subsystems on the SoC. There are various types of NoCs such as System NoC (SNoC), Config NoC (CNoC), Multimedia NoC (MMNoC).
PCB	Printed circuit board
PMIC	Power management IC is the power supply block of the chipset.
PS_HOLD	Power-supply hold signal line from the SoC to the PMIC
QCAP	Qualcomm crash analysis portal
SNoC	System network-on-chip
SPM	Subsystem power manager

Acronym or term	Definition
SMMU	System memory management unit
SS	Subsystem
TRACE32	Lauterbach TRACE32 software
TBU	Translation buffer unit. Arm® SMMU IP component.
TCM	Tightly coupled memory
TEE	Trusted execution environment
TSENS	Temperature sensor; this sensor captures the junction temperature of the chipset.
TZ	TrustZone
WD	Watchdog; a watchdog is a fixed-length counter that enables a system to recover from an unexpected hardware or software catastrophe.
WD Bark	Watchdog timeout that results in a bark interrupt and a kernel panic.
WB Bite	Watchdog timeout that occurs if a watchdog is not petted even after WD Bark, resulting in a bite interrupt in secure mode. This issue further leads to a system reset.
WPSS	Wireless local area network processor subsystem
XBL	eXtensible Boot Loader
xPU	External protection unit is a module in the Qualcomm TEE meant for protecting memory regions, addresses, and registers.

LEGAL INFORMATION

Your access to and use of this material, along with any documents, software, specifications, reference board files, drawings, diagnostics and other information contained herein (collectively this "Material"), is subject to your (including the corporation or other legal entity you represent, collectively "You" or "Your") acceptance of the terms and conditions ("Terms of Use") set forth below. If You do not agree to these Terms of Use, you may not use this Material and shall immediately destroy any copy thereof.

1) Legal Notice.

This Material is being made available to You solely for Your internal use with those products and service offerings of Qualcomm Technologies, Inc. ("**Qualcomm Technologies**"), its affiliates and/or licensors described in this Material, and shall not be used for any other purposes. If this Material is marked as "**Qualcomm Internal Use Only**", no license is granted to You herein, and You must immediately (a) destroy or return this Material to Qualcomm Technologies, and (b) report Your receipt of this Material to qualcomm.support@qti.qualcomm.com. This Material may not be altered, edited, or modified in any way without Qualcomm Technologies' prior written approval, nor may it be used for any machine learning or artificial intelligence development purpose which results, whether directly or indirectly, in the creation or development of an automated device, program, tool, algorithm, process, methodology, product and/or other output. Unauthorized use or disclosure of this Material or the information contained herein is strictly prohibited, and You agree to indemnify Qualcomm Technologies, its affiliates and licensors for any damages or losses suffered by Qualcomm Technologies, its affiliates and/or licensors for any such unauthorized uses or disclosures of this Material, in whole or part.

Qualcomm Technologies, its affiliates and/or licensors retain all rights and ownership in and to this Material. No license to any trademark, patent, copyright, mask work protection right or any other intellectual property right is either granted or implied by this Material or any information disclosed herein, including, but not limited to, any license to make, use, import or sell any product, service or technology offering embodying any of the information in this Material.

THIS MATERIAL IS BEING PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESSED, IMPLIED, STATUTORY OR OTHERWISE. TO THE MAXIMUM EXTENT PERMITTED BY LAW, QUALCOMM TECHNOLOGIES, ITS AFFILIATES AND/OR LICENSORS SPECIFICALLY DISCLAIM ALL WARRANTIES OF TITLE, MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, SATISFACTORY QUALITY, COMPLETENESS OR ACCURACY, AND ALL WARRANTIES ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. MOREOVER, NEITHER QUALCOMM TECHNOLOGIES, NOR ANY OF ITS AFFILIATES AND/OR LICENSORS, SHALL BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY EXPENSES, LOSSES, USE, OR ACTIONS HOWSOEVER INCURRED OR UNDERTAKEN BY YOU IN RELIANCE ON THIS MATERIAL.

Certain product kits, tools and other items referenced in this Material may require You to accept additional terms and conditions before accessing or using those items.

Technical data specified in this Material may be subject to U.S. and other applicable export control laws. Transmission contrary to U.S. and any other applicable law is strictly prohibited.

Nothing in this Material is an offer to sell any of the components or devices referenced herein.

This Material is subject to change without further notification.

In the event of a conflict between these Terms of Use and the *Website Terms of Use* on www.qualcomm.com, the *Qualcomm Privacy Policy* referenced on www.qualcomm.com, or other legal statements or notices found on prior pages of the Material, these Terms of Use will control. In the event of a conflict between these Terms of Use and any other agreement (written or click-through, including, without limitation any non-disclosure agreement) executed by You and Qualcomm Technologies or a Qualcomm Technologies affiliate and/or licensor with respect to Your access to and use of this Material, the other agreement will control.

These Terms of Use shall be governed by and construed and enforced in accordance with the laws of the State of California, excluding the U.N. Convention on International Sale of Goods, without regard to conflict of laws principles. Any dispute, claim or controversy arising out of or relating to these Terms of Use, or the breach or validity hereof, shall be adjudicated only by a court of competent jurisdiction in the county of San Diego, State of California, and You hereby consent to the personal jurisdiction of such courts for that purpose.

2) Trademark and Product Attribution Statements.

Qualcomm is a trademark or registered trademark of Qualcomm Incorporated. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the U.S. and/or elsewhere. The Bluetooth® word mark is a registered trademark owned by Bluetooth SIG, Inc. Other product and brand names referenced in this Material may be trademarks or registered trademarks of their respective owners.

Snapdragon and Qualcomm branded products referenced in this Material are products of Qualcomm Technologies, Inc. and/or its subsidiaries. Qualcomm patented technologies are licensed by Qualcomm Incorporated.