DEBUGGING EMBEDDED LINUX KERNEL & DRIVER

Kishore Kumar

KERNEL MASTERS www.kernelmasters.org

Contents

Revision History

1.0	Basic Version created By Kishore kumar Boddu

1. KDB & KGDB

KDB- Kernel Debugger:

- Kdb is an instruction-level debugger used for debugging kernel code and device drivers.
- Before you can use it, you need to enable configuration options your kernel sources with kdb support and recompile the kernel.

KGDB- Kernel GNU Debugger:

- Kgdb is a source-level debugger.
- It is easier to use than kdb because you don't have to spend time correlating assembly code with your sources.
- However it's more difficult to set up because an additional machine is needed to front-end the debugging.
- gdb runs on the host machine, while the kgdb-patched kernel runs on the target hardware.
- The host and the target are connected via a serial null-modem cable.

Setup Kernel Debug Environment for KDB & KGDB

Step 1: Enable KGDB/KDB options

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig
```

CONFIG_DEBUG_INFO: Compiles the kernel with debug symbols (-g option)

```
CONFIG_DEBUG_INFO=y
---> Kernel Hacking
        -- > Compile-time checks and compiler options
                -- > Compile the kernel with debug info
```

CONFIG_KGDB: enables support for KGDB

```
CONFIG_KGDB=y
---> Kernel Hacking
        ---> KGDB: Kernel Debugger
```

CONFIG_KGDB_SERIAL_CONSOLE: Enables KGDB communication I/O driver over the serial port

```
CONFIG_KGDB_SERIAL_CONSOLE=y
---> Kernel Hacking
        ---> KGDB: Kernel Debugger
                ---> KGDB: use kgdb over the serial console
```

```
CONFIG_KGDB_KDB=y
---> Kernel Hacking
        ---> KGDB: Kernel Debugger
                ---> KGDB_KDB: include kdb frontend for kgdb
```

```
CONFIG_KDB_KEYBOARD=y
---> Kernel Hacking
        ---> KGDB: Kernel Debugger
                ---> KGDB_KDB: include kdb frontend for kgdb
```

CONFIG_FRAME_POINTER: Helps to produce more reliable stack traces

CONFIG_MAGIC_SYSRQ: Enables magic sysrq key functionality to put the kernel in debug mode

```
---> Kernel Hacking
          ---> KGDB: Magic SysRq key
```

CONFIG_DEBUG_RODATA_TEST not set

```
          ---> Kernel Hacking
                    ---> Memory Debugging
                              ---> Testcase for the marking rodata read-only (disable this option)
```

CONFIG_STRICT_KERNEL_RWX is not set

Step 2: Kernel configuration & build

```
$ ./km-bbb-kernel-build.sh
```

Step 3: Kernel Installation

```
$ ./km-bbb-kernel-install.sh
```

Step 4: Setup kernel bootargs in u-boot

Boot any one of the mode MMC0, MMC1 & TFTP.

setenv bootargs console=tty0 console=${console} root=/dev/mmcblk1p1
rootfstype=${mmcrootfstype} ${cmdline}; root=/dev/mmcblk1p1 console=ttyO0,115200n8
rodata=off nokaslr kgdb=ttyO0,115200 kgdboc=ttyO0,115200n8 kgdbwait rootwait;

Kernel Address Space Randomization (KASLR)

setenv bootargs rootfstype=${mmcrootfstype} root=/dev/mmcblk0p1
console=ttyS0,115200n8 rodata=off nokaslr kgdb=ttyS0,115200
kgdboc=ttyS0,115200n8 kgdbwait rootwait;

KDB Testing:

Choose KDB/KGDB Enabled Kernel at bootloader:

Power on the board, choose "Boot with KGDB/KDB" option in "KM boot menu" at u-boot prompt.
Bydefault kernel enter to KDB mode.
Kernel assign a break point after serial initialization and wait for user input.

```
kdb> help (list of kdb commands)
kdb> go (continue kernel boot process)
```

Login as user until you see prompt.

Entering a Kernel Debugger:

Case 1: User Entering a Kernel Debugger [KDB/KGDB]:

Enter supervisor mode.

```
$ sudo su
```

assign a break point to kernel.

```
$ echo g > /proc/sysrq-trigger
```

Kernel enter to kdgb break point and shows kdb prompt.

```
kdb>
```

Case 2: Whenever Kernel panic is occurs by default Enter KDB prompt.

kdb commands

```
[0]kdb> help
Command       Usage             Description
----------------------------------------------------------
md          <vaddr>          Display Memory Contents, also mdWcN, e.g1
mdr         <vaddr> <bytes>   Display Raw Memory
mdp         <paddr> <bytes>   Display Physical Memory
mds         <vaddr>          Display Memory Symbolically
mm          <vaddr> <contents>  Modify Memory Contents
go          [<vaddr>]         Continue Execution
rd                            Display Registers
rm          <reg> <contents>   Modify Registers
ef          <vaddr>          Display exception frame
bt          [<vaddr>]         Stack traceback
btp         <pid>             Display stack for process <pid>
bta         [D|R|S|T|C|Z|E|U|I|M|A]
                             Backtrace all processes matching state fg
btc                           Backtrace current process on each cpu
btt         <vaddr>          Backtrace process given its struct task s
env                           Show environment variables
set                           Set environment variables
help                          Display Help Message
?                             Display Help Message
cpu         <cpunum>          Switch to new cpu
kgdb                          Enter kgdb mode
ps          [<flags>|A]       Display active task list
pid         <pidnum>          Switch to another task
reboot                        Reboot the machine immediately
lsmod                         List loaded kernel modules
sr          <key>             Magic SysRq key
dmesg       [lines]           Display syslog buffer
defcmd      name "usage" "help" Define a set of commands, down to endefcd
kill        <-signal> <pid>   Send a signal to a process
summary                       Summarize the system
per_cpu     <sym> [<bytes>] [<cpu>]
                             Display per_cpu variables
grephelp                      Display help on | grep
bp          [<vaddr>]         Set/Display breakpoints
bl          [<vaddr>]         Display breakpoints
bc          <bpnum>           Clear Breakpoint
be          <bpnum>           Enable Breakpoint
bd          <bpnum>           Disable Breakpoint
ss                            Single Step
dumpcommon                    Common kdb debugging
dumpall                       First line debugging
dumpcpu                       Same as dumpall but only tasks on cpus
ftdump      [skip_#lines] [cpu] Dump ftrace log
```

KGDB Debugging (source level debugging):

Whenever kernel crash, panic occurs by default enter in to KDB prompt. Switch from KDB to KGDB.

Step 1: Connect target board remote using ssh.

Run the below command in host machine and connect target board remote. Provide target board user name and IP address to the below command.

```
$ ssh <user_name>@<IP_Address>
```

Once remote login target board and run the below command to assign a break point to the kernel.

```
$ sudo su
# echo g > /proc/sysrq-trigger
```

Step 2: Switch KDB to KGDB

Enter kgdb command in kdb prompt.

```
kdb> kgdb <ENTER>
```

You will now see "Entering please attach debugger or use $D#44 or $3#33"

Close serial port terminal (minicom or teraterm) and open terminal in host machine.
Assuming serial port device name is ttyUSB0

Step 3: Remote Debugging with GDB

`$arm-linux-gnueabihf-gdb vmlinn` (this command shows gdb prompt it is called KGDB because image name is "vmlinux")

```
        (gdb) set serial baud 115200
        (gdb) target remote /dev/ttyUSB0
        (gdb) bt (shows back trace of kgdb_breakpoint())
        (gdb) b omap_gpio_get
        (gdb) c (continue)
```

Step 4: Read gpio pin value from sysfs.

```
$ cd /sys/class/gpio/gpio9
$ cat value
```

Than you will see the gdb prompt with break point triggered

2. Module debugging (gpio_input.c)

Step 1: Compile module with debugging symbols

Add KBUILD_CFLAGS += -g flag in top most Makefile in Kernel source code.

Step 2: Copy module source code .ko files in to host

```
$ scp char.ko <username>@<IPAddress>:~/
$ sudo insmod char.ko
```

Step 3: Identify Section addresses

```
$ sudo su // Should be in root user
# cd /sys/modules/char/sections
# ls -la
# cat .text .data .bss
```

Step 4: Assign a breakpoint in Kernel

Host Machine:

```
$  sudo su
# echo g > /proc/sysrq-trigger
```

Step 5: Load module symbols

gdb prompt:

```
(gdb) add-symbol-file /home/km/debug/kernel/gpio/gpio-input.ko 0xbf428000 -s .data 0xbf42a000 -
s .bss 0xbf42a300
add symbol table from file "/home/km/debug/kernel/gpio/gpio-input.ko" at
        .text_addr = 0xbf428000
        .data_addr = 0xbf42a000
        .bss_addr = 0xbf42a300
(y or n) y
Reading symbols from /home/km/debug/kernel/gpio/gpio-input.ko...done.

(gdb) disass example_read
Dump of assembler code for function example_read:
   0xbf428000 <+0>:     push    {r4, r5, r6, r7, r8, lr}
   0xbf428004 <+4>:     sub     sp, sp, #40         ; 0x28
   0xbf428008 <+8>:     ldr     r7, [pc, #216]      ; 0xbf4280e8 <example_read+232>
   0xbf42800c <+12>:    ldr     r0, [pc, #216]      ; 0xbf4280ec <example_read+236>
   0xbf428010 <+16>:    mov     r4, r2
   0xbf428014 <+20>:    ldr     r3, [r7]
   0xbf428018 <+24>:    ldr     r0, [r0]
   0xbf42801c <+28>:    mov     r6, r1
   0xbf428020 <+32>:    str     r3, [sp, #36]       ; 0x24
   0xbf428024 <+36>:    bl      0xc0460d88 <gpio_to_desc>
   0xbf428028 <+40>:    bl      0xc045fa90 <gpiod_get_raw_value>
   0xbf42802c <+44>:    ldr     r2, [pc, #188]      ; 0xbf4280f0 <example_read+240>
   0xbf428030 <+48>:    mov     r1, #32
   0xbf428034 <+52>:    mov     r3, r0
   0xbf428038 <+56>:    add     r0, sp, #4
   0xbf42803c <+60>:      bl          0xc081e340 <snprintf>
--Type <RET> for more, q to quit, c to continue without paging--q
Quit

(gdb) b example_read
Breakpoint 1 at 0xbf428000: file /home/km/debug/kernel/gpio/gpio-input.c, line 87.
(gdb) b omap_gpio_get
Breakpoint 2 at 0xc046702c: file drivers/gpio/gpio-omap.c, line 1009.
(gdb) c
Continuing.
```

Step 6: read gpio_input device file from host side than automatically breakpoint comes to target side.

Host Machine:

```
$  sudo su
# cat /dev/gpio_input
```

```
Thread 96 hit Breakpoint 1, example_read (filp=0xdb7727c0, buffer=0xb6982000 <error: Cannot
access memory at address 0xb6982000>, length=131072, offset=0xdb75ff78)
    at /home/km/debug/kernel/gpio/gpio-input.c:87
87        {

(gdb) i b
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0xbf428000 in example_read at /home/km/debug/kernel/gpio/gpio-
input.c:87
             breakpoint already hit 1 time
2       breakpoint     keep y   0xc046702c in omap_gpio_get at drivers/gpio/gpio-omap.c:1009

(gdb) bt
#0  example_read (filp=0xdb7727c0, buffer=0xb6982000 <error: Cannot access memory at address
0xb6982000>, length=131072, offset=0xdb75ff78)
    at /home/km/debug/kernel/gpio/gpio-input.c:87
#1  0xc01f4348 in vfs_read (pos=<optimized out>, count=<optimized out>, buf=<optimized out>,
file=<optimized out>) at fs/read_write.c:452
#2  vfs_read (file=0xdb7727c0, buf=0xb6982000 <error: Cannot access memory at address
0xb6982000>, count=131072, pos=0xdb75ff78) at fs/read_write.c:437
#3  0xc01f4748 in ksys_read (fd=<optimized out>, buf=0xb6982000 <error: Cannot access memory at
address 0xb6982000>, count=131072) at fs/read_write.c:605
#4  0xc0009000 in __idmap_text_end ()
Backtrace stopped: previous frame identical to this frame (corrupt stack?)

Thread 96 hit Breakpoint 1, example_read (filp=0xdb7727c0, buffer=0xb6982000 "1",
length=131072, offset=0xdb75ff78) at /home/km/debug/kernel/gpio/gpio-input.c:87
87        {
(gdb) c
Continuing.
```

```
Thread 96 hit Breakpoint 2, omap_gpio_get (chip=0xde18ecd4, offset=11) at drivers/gpio/gpio-omap.c:1009
1009            bank = gpiochip_get_data(chip);
(gdb) c
Continuing.

Thread 96 hit Breakpoint 1, example_read (filp=0xdb7727c0, buffer=0xb6982000 "1", length=131072, offset=0xdb75ff78) at /home/km/debug/kernel/gpio/gpio-input.c:87
87      {
(gdb) c
Continuing.

Thread 96 hit Breakpoint 2, omap_gpio_get (chip=0xde18ecd4, offset=11) at drivers/gpio/gpio-omap.c:1009
1009            bank = gpiochip_get_data(chip);
(gdb) c
Continuing.

Thread 96 hit Breakpoint 1, example_read (filp=0xdb7727c0, buffer=0xb6982000 "1", length=131072, offset=0xdb75ff78) at /home/km/debug/kernel/gpio/gpio-input.c:87
87      {
(gdb) n
80        device_destroy(example_class, example_dev);
(gdb) n
81        class_destroy(example_class);
(gdb) n
90        snprintf(chaine, 32, "%d", gpio_get_value(gpio_in));
(gdb) n

Thread 96 hit Breakpoint 2, omap_gpio_get (chip=0xde18ecd4, offset=11) at drivers/gpio/gpio-omap.c:1009
1009            bank = gpiochip_get_data(chip);
(gdb) n
1011            if (omap_gpio_is_input(bank, offset))
(gdb) n
1012                    return omap_get_gpio_datain(bank, offset);
(gdb) n
1014                    return omap_get_gpio_dataout(bank, offset);
(gdb) n
gpiod_get_raw_value_commit (desc=0xde18eab0) at drivers/gpio/gpiolib.c:2832
2832            value = value < 0 ? value : !!value;
(gdb) p value
$10 = <optimized out>
(gdb) n
2833            trace_gpio_value(desc_to_gpio(desc), 1, value);
(gdb) n
warning: Error removing breakpoint 0
[New Thread 752]
```

3. Linux Kernel Crash dump

Install crass dump tools:

```
$ sudo apt install kexec-tools
$ sudo apt-get install kdump-tools
```

Using kdump and kexec analyze the kernel coredump files.

kexec:

kexec is directly boot into a new kernel without going through BIOS.
Kexec is a fastboot mechanism that allows booting a Linux kernel from the context of an already running kernel without going through BIOS.
BIOS can be very time consuming, especially on big servers with numerous peripherals.
This can save a lot of time for developers who end up booting a machine numerous times kexec is a system call that enables you to load and boot into another kernel from the currently running kernel.
kexec performs the function of the boot loader from within the kernel.
The primary difference between a standard system boot and a kexec boot is that the hardware initialization normally performed by the BIOS or firmware (depending on architecture) is not performed during a kexec boot. This has the effect of reducing the time required for a reboot

kdump:

Kdump is a new kernel crash dumping mechanism and is very reliable.
The crash dump is captured from the context of a freshly booted kernel and not from the context of the crashed kernel.
Kdump uses Kexec to boot into a second kernel whenever the system crashes.
This second kernel, often called a crash or a capture kernel, boots with very little memory and captures the dump image.
The first kernel reserves a section of memory that the second kernel uses to boot.
Kexec enables booting the capture kernel without going through BIOS hence the contents of the first kernel's memory are preserved, which is essentially the kernel crash dump.

kdump is a feature of the Linux kernel that creates crash dumps in the event of a kernel crash.
When triggered, kdump exports a memory image (also known as vmcore) that can be analyzed for he purposes of debugging and determining the cause of a crash.

Enable the below kernel configuration options & compile:

System kernel config options:
1) Enable "kexec system call" in "Processor type and features."
This parameter tells the system to use Kexec to skip BIOS and boot (new) kernels. It is critical for the functionality of Kdump.

```
  CONFIG_KEXEC=y
         Boot options  --->
                              [*] Kexec system call (EXPERIMENTAL)
```

2) Enable "sysfs file system support" in "Filesystem" -> "Pseudo filesystems." This is usually enabled by default

```
  CONFIG_SYSFS=y
```

3) Enable "Compile the kernel with debug info" in "Kernel hacking."
  This causes the kernel to be built with debug symbols. The dump analysis tools require a vmlinux with debug symbols in order to read and analyze a dump file

```
CONFIG_DEBUG_INFO=y
```

Dump-capture kernel config options (Arch Independent):

1) Enable kernel crash dumps: Crash dumps need to be enabled. Without this option, Kdump will be useless.

```
  CONFIG_CRASH_DUMP=y
      Boot options  --->
                              Build kdump crash kernel (EXPERIMENTAL)
```

2) Enable "/proc/vmcore support" under "Filesystems" -> "Pseudo filesystems".

```
  CONFIG_PROC_VMCORE=y
  (CONFIG_PROC_VMCORE is set by default when CONFIG_CRASH_DUMP is selected.)
   -> File systems
                      -> Pseudo filesystems
                               -> /proc file system support (PROC_FS [=y])
```

Dump-capture kernel config options (Arch Dependent, i386 and x86_64)

1) If one wants to build and use a relocatable kernel,
  Enable "Build a relocatable kernel" support under "Processor type and features"

```
CONFIG_RELOCATABLE=y
```

2) Optional: Disable Symmetric Multi-Processing (SMP) support: Kdump can only work with a single processor.

If you have only a single processor or you run your machine with SMP support disabled, you can safely set this parameter to (n).

```
CONFIG_SMP=n
KDUMP_COMMANDLINE_APPEND="maxcpus=1"
```

(If CONFIG_SMP=y, then specify maxcpus=1 on the kernel command line when loading the dump-capture kernel, see section "Load the Dump-capture Kernel".)

3) Use a suitable value for "Physical address where the kernel is loaded" (under "Processor type and features").

  This only appears when "kernel crash dumps" is enabled. A suitable value depends upon whether kernel is relocatable or not.

  If you are using a relocatable kernel use CONFIG_PHYSICAL_START=0x100000
  This will compile the kernel for physical address 1MB, but given the fact kernel is relocatable, it can be run from any physical address hence kexec boot loader will load it in memory region reserved for dump-capture kernel.

Otherwise it should be the start of memory region reserved for second kernel using boot parameter "crashkernel=Y@X".

Here X is start of memory region reserved for dump-capture kernel.
  Generally X is 16MB (0x1000000). So you can set CONFIG_PHYSICAL_START=0x1000000

5) Make and install the kernel and its modules.

Dump-capture kernel config options (Arch Dependent, arm)

1) To use a relocatable kernel,
   Enable "AUTO_ZRELADDR" support under "Boot" options:
   AUTO_ZRELADDR=y

crash kernel recommendations:

For memory between 2G through 4G, reserve 320M
For memory between 4G through 32G, reserve 512M
For memory between 32G through 64G, reserve 1024M
For memory between 64G through 128G, reserve 2048M
For memory above 128G, reserve 4096M

add "crashkernel=256M" in /boot/grub/grub.cfg file.

Example:

```
Command line: BOOT_IMAGE=/boot/vmlinuz-4.4.88 root=UUID=aab9b05c-202b-42c6-8be6-
c90e9d6af8e8 ro quiet splash crashkernel=256M vt.handoff=7
```

```
$ kdump-config status
$ kdump-config show
$ kdump-config load
$ cat /proc/sys/kernel/sysrq
$ sudo sysctl -w kernel.sysrq=1
```

```
$ sudo kexec -l /boot/vmlinuz-4.19.132 --append="BOOT_IMAGE=/boot/vmlinuz-4.19.132
root=UUID=f51c4fff-68ae-4bb9-86dd-5d8716be2821 ro quiet splash" --initrd=/boot/initrd.img-
4.19.132
```

```
$ sudo kexec -l /boot/vmlinuz-4.19.94-Kernel-Masters-g001db1705-dirty --
append="console=ttyO0,115200n8 root=/dev/mmcblk1p1 rootfstype=ext4 rootwait"
```

```
$ sudo kexec -e
```

```
kexec --type zImage -p /boot/vmlinuz-4.19.94-Kernel-Masters-gb3f1becdc-dirty --
initrd=/var/lib/kdump/initrd.img-4.19.94-Kernel-Masters-g001db1705-dirty --
dtb=/boot/dtbs/4.19.94-Kernel-Masters-gb3f1becdc-dirty/am335x-boneblack.dtb --
append="console=ttyO0,115200n8 root=/dev/mmcblk1p1 rootfstype=ext4 rootwait
crashkernel=64B maxcpus=1 reset_devices"
```

```
kexec --type zImage -p  --command-line="console=ttyO0,115200n8 root=/dev/mmcblk1p1
rootfstype=ext4 rootwait crashkernel=64B maxcpus=1 reset_devices" /boot/vmlinuz-4.19.94-Kernel-
Masters-gb3f1becdc-dirty --initrd=/var/lib/kdump/initrd.img-4.19.94-Kernel-Masters-g001db1705-
dirty --dtb=/boot/dtbs/4.19.94-Kernel-Masters-gb3f1becdc-dirty/am335x-boneblack.dtb
```

```
  /sbin/kexec -u /boot/vmlinuz-4.19.94-Kernel-Masters-gb3f1becdc-dirty --dtb=/boot/dtbs/4.19.94-
Kernel-Masters-gb3f1becdc-dirty/am335x-boneblack.dtb --command-line="console=ttyO0,115200n8
root=/dev/mmcblk1p1 rootfstype=ext4 rootwait maxcpus=1 reset_devices nr_cpus=1
systemd.unit=kdump-tools.service irqpoll nousb ata_piix.prefer_ms_hyperv=0" --
initrd=/var/lib/kdump/initrd.img /var/lib/kdump/vmlinuz
```

```
kexec --type=zImage -p /boot/vmlinuz-4.19.94-Kernel-Masters-gb3f1becdc-dirty --
dtb=/boot/dtbs/4.19.94-Kernel-Masters-gb3f1becdc-dirty/am335x-boneblack.dtb --
append="console=ttyO0,115200n8 root=/dev/mmcblk1p1 rootfstype=ext4 rootwait maxcpus=1
reset_devices"

Usage: kexec [OPTION]... [kernel]
Directly reboot into a new kernel

 -h, --help         Print this help.
 -v, --version      Print the version of kexec.
 -f, --force        Force an immediate kexec,
                    don't call shutdown.
 -i, --no-checks    Fast reboot, no memory integrity checks.
 -x, --no-ifdown    Don't bring down network interfaces.
 -y, --no-sync      Don't sync filesystems before kexec.
 -l, --load         Load the new kernel into the
                    current kernel.
 -p, --load-panic   Load the new kernel for use on panic.
 -u, --unload       Unload the current kexec target kernel.
                    If capture kernel is being unloaded
                    specify -p with -u.
 -e, --exec         Execute a currently loaded kernel.
 -t, --type=TYPE    Specify the new kernel is of this type.
     --mem-min=<addr> Specify the lowest memory address to
                    load code into.
     --mem-max=<addr> Specify the highest memory address to
                    load code into.
     --reuseinitrd  Reuse initrd from first boot.
     --print-ckr-size Print crash kernel region size.
     --load-preserve-context Load the new kernel and preserve
                    context of current kernel during kexec.
     --load-jump-back-helper Load a helper image to jump back
                    to original kernel.
     --entry=<addr> Specify jump back address.
                    (0 means it's not jump back or
                    preserve context)
                    to original kernel.
 -s, --kexec-file-syscall Use file based syscall for kexec operation
 -c, --kexec-syscall  Use the kexec_load syscall for for compatibility
                    with systems that don't support -s (default)
 -a, --kexec-syscall-auto  Use file based syscall for kexec and fall
                    back to the compatibility syscall when file based
                    syscall is not supported or the kernel did not
                    understand the image
 -d, --debug        Enable debugging to help spot a failure.
 -S, --status       Return 0 if the type (by default crash) is loaded.

Supported kernel file types and options:
uImage
     --command-line=STRING Set the kernel command line to STRING.
     --append=STRING       Set the kernel command line to STRING.
```

```
    --initrd=FILE        Use FILE as the kernel's initial ramdisk.
    --ramdisk=FILE       Use FILE as the kernel's initial ramdisk.
    --dtb=FILE           Use FILE as the fdt blob.
    --atags              Use ATAGs instead of device-tree.
    --page-offset=PAGE_OFFSET
                         Set PAGE_OFFSET of crash dump vmcore
zImage
    --command-line=STRING Set the kernel command line to STRING.
    --append=STRING       Set the kernel command line to STRING.
    --initrd=FILE        Use FILE as the kernel's initial ramdisk.
    --ramdisk=FILE       Use FILE as the kernel's initial ramdisk.
    --dtb=FILE           Use FILE as the fdt blob.
    --atags              Use ATAGs instead of device-tree.
    --page-offset=PAGE_OFFSET
                         Set PAGE_OFFSET of crash dump vmcore
Architecture options:
    --image-size=<size>
            Specify the assumed total image size of
            the kernel that is about to be loaded,
            including the .bss section, as reported
            by 'arm-linux-size vmlinux'. If not
            specified, this value is implicitly set
            to the compressed images size * 4.
    --dt-no-old-root
            do not reuse old kernel root= param.
            while creating flatten device tree.
```

Core Dump Analyse:

1. using gdb tool

```
$ sudo gdb KM_GIT/linux-4.4.88/vmlinux /var/crash/201712022302/vmcore.201712022302
[sudo] password for kernel:
GNU gdb (Ubuntu 7.7.1-0ubuntu5~14.04.3) 7.7.1
Copyright (C) 2014 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/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from KM_GIT/linux-4.4.88/vmlinux...done.
[New LWP 2451]
[New LWP 1120]
[New process 1]
```

```
[New LWP 2142]
#0  sysrq_handle_crash (key=99) at drivers/tty/sysrq.c:138
138                     *killer = 1;
warning: File "/home/kernel/KM_GIT/linux-4.4.88/scripts/gdb/vmlinux-gdb.py" auto-loading has
been declined by your `auto-load safe-path' set to "$debugdir:$datadir/auto-load".
To enable execution of this file add
            add-auto-load-safe-path /home/kernel/KM_GIT/linux-4.4.88/scripts/gdb/vmlinux-gdb.py
line to your configuration file "/home/kernel/.gdbinit".
To completely disable this security protection add
            set auto-load safe-path /
line to your configuration file "/home/kernel/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
            info "(gdb)Auto-loading safe path"
(gdb) bt
#0  sysrq_handle_crash (key=99) at drivers/tty/sysrq.c:138
#1  0xffffffff814c28c9 in __handle_sysrq (key=99, check_mask=false) at drivers/tty/sysrq.c:545
#2  0xffffffff814c2d18 in write_sysrq_trigger (file=<optimized out>, buf=<optimized out>, count=1,
ppos=<optimized out>)
   at drivers/tty/sysrq.c:1091
#3  0xffffffff8126828d in proc_reg_write (file=<optimized out>, buf=<optimized out>,
count=<optimized out>,
   ppos=<optimized out>) at fs/proc/inode.c:216
#4  0xffffffff811fd558 in __vfs_write (file=0xffff8802411b8700, p=<optimized out>,
count=<optimized out>,
   pos=0x0 <irq_stack_union>) at fs/read_write.c:489
#5  0xffffffff811fdc12 in vfs_write (file=0x63 <irq_stack_union+99>, buf=0xffff88024f40dd98 "",
count=<optimized out>,
   pos=0xffff88024307bf20) at fs/read_write.c:538
#6  0xffffffff811fe936 in SYSC_write (count=<optimized out>, buf=<optimized out>, fd=<optimized
out>)
   at fs/read_write.c:585
#7  SyS_write (fd=<optimized out>, buf=15049736, count=2) at fs/read_write.c:577
#8  0xffffffff817e6e36 in entry_SYSCALL_64_fastpath () at arch/x86/entry/entry_64.S:185
#9  0x0000000000e8f608 in ?? ()
#10 0x0000000000000004 in irq_stack_union ()
```

2. Using crash tool

```
$ cd /var/crash/
$ ls
201902261006  kexec_cmd
$ cd 201902261006/

$ sudo apt source linux-image-`uname -r`

$ crash <debug kernel> <crash dump>

$ sudo crash KM_GIT/linux-4.4.88/vmlinux /var/crash/201712022302/vmcore.201712022302

crash 7.0.3
```

```
Copyright (C) 2002-2013  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  NEC Corporation
Copyright (C) 1999, 2002, 2007  Silicon Graphics, Inc.
Copyright (C) 1999, 2000, 2001, 2002  Mission Critical Linux, 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.

GNU gdb (GDB) 7.6
Copyright (C) 2013 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-unknown-linux-gnu"...


      KERNEL: KM_GIT/linux-4.4.88/vmlinux
    DUMPFILE: /var/crash/201712022302/vmcore.201712022302
        CPUS: 4
        DATE: Thu Jan  1 05:30:00 1970
      UPTIME: 00:05:51
LOAD AVERAGE: 0.36, 0.52, 0.29
       TASKS: 367
    NODENAME: KernelMasters
     RELEASE: 4.4.88
     VERSION: #1 SMP Fri Dec 1 22:09:18 IST 2017
     MACHINE: x86_64  (2711 Mhz)
      MEMORY: 7.9 GB
       PANIC: "Oops: 0002 [#1] SMP " (check log for details)
         PID: 2451
     COMMAND: "bash"
        TASK: ffff880243510000  [THREAD_INFO: ffff880243078000]
         CPU: 0
       STATE: TASK_RUNNING (PANIC)

crash>
crash> help

*          files        mach         repeat       timer
alias      foreach      mod          runq         tree
ascii      fuser        mount        search       union
bt         gdb          net          set          vm
btop       help         p            sig          vtop
dev        ipcs         ps           struct       waitq
dis        irq          pte          swap         whatis
```

```
eval        kmem        ptob        sym         wr
exit        list        ptov        sys         q
extend      log         rd          task

crash version: 7.0.3    gdb version: 7.6
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> ps
   PID    PPID  CPU       TASK        ST  %MEM     VSZ    RSS  COMM
      0      0   0  ffffffff81c13500  RU   0.0       0      0  [swapper/0]
      0      0   1  ffff880245699e00  RU   0.0       0      0  [swapper/1]
>     0      0   2  ffff88024569ad00  RU   0.0       0      0  [swapper/2]
      0      0   3  ffff88024569bc00  RU   0.0       0      0  [swapper/3]
      1      0   3  ffff880245b58000  IN   0.0   34060   4576  init

crash> bt
PID: 2451   TASK: ffff880243510000  CPU: 0   COMMAND: "bash"
 #0 [ffff88024307bac0] machine_kexec at ffffffff8105919c
 #1 [ffff88024307bb18] crash_kexec at ffffffff81106153
 #2 [ffff88024307bbe0] oops_end at ffffffff8101aae9
 #3 [ffff88024307bc08] no_context at ffffffff81066f3d
 #4 [ffff88024307bc60] __bad_area_nosemaphore at ffffffff810672b9
 #5 [ffff88024307bca8] bad_area_nosemaphore at ffffffff810673d3
 #6 [ffff88024307bcb8] __do_page_fault at ffffffff810679c0
 #7 [ffff88024307bd10] do_page_fault at ffffffff81067d82
 #8 [ffff88024307bd30] page_fault at ffffffff817e8fb8
    [exception RIP: sysrq_handle_crash+22]
    RIP: ffffffff814c2116  RSP: ffff88024307bde0  RFLAGS: 00010296
    RAX: 000000000000000f  RBX: ffffffff81cbb800  RCX: 0000000000000000
    RDX: 0000000000000001  RSI: ffff88024f40dd98  RDI: 0000000000000063
    RBP: ffff88024307bde0   R8: ffffffff81f19ed4   R9: 0000000000000030
    R10: ffffffff81f0937c  R11: 0000000000000358  R12: 0000000000000063
    R13: 0000000000000000  R14: 0000000000000004  R15: 0000000000000000
    ORIG_RAX: ffffffffffffffff  CS: 0010  SS: 0018
 #9 [ffff88024307bde8] __handle_sysrq at ffffffff814c28c9
#10 [ffff88024307be18] write_sysrq_trigger at ffffffff814c2d18
#11 [ffff88024307be30] proc_reg_write at ffffffff8126828d
#12 [ffff88024307be50] __vfs_write at ffffffff811fd558
#13 [ffff88024307bed0] vfs_write at ffffffff811fdc12
#14 [ffff88024307bf10] sys_write at ffffffff811fe936
#15 [ffff88024307bf50] entry_SYSCALL_64_fastpath at ffffffff817e6e36
    RIP: 00007f6f67b15390  RSP: 00007ffca2f8acd8  RFLAGS: 00000246
    RAX: ffffffffffffffda  RBX: 00000000004b7ddd  RCX: 00007f6f67b15390
    RDX: 0000000000000002  RSI: 0000000000e5a408  RDI: 0000000000000001
    RBP: 00000000004b7ddd   R8: 000000000000000a   R9: 00007f6f68423740
    R10: 00007f6f67de76a0  R11: 0000000000000246  R12: 0000000000e74e88
    R13: 0000000000000004  R14: 0000000000000004  R15: 0000000000e8f608
    ORIG_RAX: 0000000000000001  CS: 0033  SS: 002b
crash>
crash>
crash> kmem -i
              PAGES        TOTAL      PERCENTAGE
 TOTAL MEM  1944950       7.4 GB         ----
      FREE  1641126       6.3 GB   84% of TOTAL MEM
      USED   303824       1.2 GB   15% of TOTAL MEM
    SHARED    43821     171.2 MB    2% of TOTAL MEM
   BUFFERS    15850      61.9 MB    0% of TOTAL MEM
    CACHED   136661     533.8 MB    7% of TOTAL MEM
      SLAB    13080      51.1 MB    0% of TOTAL MEM

TOTAL SWAP  3999999      15.3 GB         ----
 SWAP USED        0            0    0% of TOTAL SWAP
 SWAP FREE  3999999      15.3 GB  100% of TOTAL SWAP

crash> irq
 IRQ   IRQ_DESC/_DATA      IRQACTION      NAME
  0   ffff88024682e800  ffffffff81c19a40  "timer"
  1   ffff88024682ea00  ffff88022c002f80  "i8042"
  2   ffff88024682ec00      (unused)
  3   ffff88024682ee00      (unused)
  4   ffff88024682f000      (unused)
  5   ffff88024682f200      (unused)
  6   ffff88024682f400      (unused)
  7   ffff88024682f600      (unused)
  8   ffff88024682f800  ffff88022c05c180  "rtc0"
  9   ffff88024682fa00  ffff88024516c100  "acpi"
 10   ffff88024682fc00      (unused)
 11   ffff88024682fe00      (unused)
 12   ffff880246888000  ffff88022c002f00  "i8042"
 13   ffff880246888200      (unused)
 14   ffff880246888400  ffff8800a2cc7b00  "INT344B:00"
 15   ffff880246888600      (unused)
 16   ffff880245363800  ffff88009906a380  "idma64.0"
                        ffff8800a6d07b00  "i2c_designware.0"
 17   ffff88022cdffe00  ffff88009906a980  "idma64.1"
```

3. apport-retrace tool:

```
$ apport-retrace --stdout --rebuild-package-info /var/crash/linux-image*.crash
```