Analyzing a kernel Oops on the AArch64 (arm64)

Target: AArch64 Raspberry Pi 4 Model B
Broadcom BCM2711 SoC
ARM Cortex-A72
Build via Yocto 5.0

Oops try code: LKD book github repo:
ch7/oops_tryv1

The Oops

```
[ 1825.203166] oops_tryv1:try_oops_init():37: Lets Oops!
               Now attempting to write something to the NULL address 0x0000000000000000
[ 1825.216268] Unable to handle kernel NULL pointer dereference at virtual address 0000000000000000
[ 1825.225198] Mem abort info:
[ 1825.228024]   ESR = 0x0000000096000046
[ 1825.231830]   EC = 0x25: DABT (current EL), IL = 32 bits
[ 1825.237218]   SET = 0, FnV = 0
[ 1825.240356]   EA = 0, S1PTW = 0
[ 1825.243567]   FSC = 0x06: level 2 translation fault
[ 1825.248533] Data abort info:
[ 1825.251445]   ISV = 0, ISS = 0x00000046, ISS2 = 0x00000000
[ 1825.257056]   CM = 0, WnR = 1, TnD = 0, TagAccess = 0
[ 1825.262182]   GCS = 0, Overlay = 0, DirtyBit = 0, Xs = 0
[ 1825.267569] user pgtable: 4k pages, 39-bit VAs, pgdp=00000000481a1000
[ 1825.274103] [0000000000000000] pgd=0800000045835003, p4d=0800000045835003, pud=0800000045835003, pmd=0000000000000000
[ 1825.284882] Internal error: Oops: 0000000096000046 [#1] PREEMPT SMP
[ 1825.291236] Modules linked in: oops_tryv1(O+) brcmfmac_wcc rpivid_hevc(C) bcm2835_codec(C) bcm2835_v4l2(C) bcm2835_isp(C) v4l2_mem2mem hci_uart bcm2835_mmal_vchiq(C) btbcm videobuf2_vmalloc videobuf2_dma_contig v3d videobuf2_memops gpu_sched brcmfmac bluetooth videobuf2_v4l2 videodev drm_shmem_helper brcmutil videobuf2_common ecdh_generic ecc raspberrypi_hwmon mc i2c_brcmstb vc_sm_cma(C) snd_bcm2835(C) raspberrypi_gpiomem uio_pdrv_genirq uio sch_fq_codel fuse nfnetlink ipv6 [last unloaded: printk_loglvl(O)]
[ 1825.336885] CPU: 3 PID: 4919 Comm: insmod Tainted: G         C O       6.6.22-kenix #1
[ 1825.344913] Hardware name: Raspberry Pi 4 Model B Rev 1.4 (DT)
[ 1825.350823] pstate: 60000005 (nZCv daif -PAN -UAO -TCO -DIT -SSBS BTYPE=--)
[ 1825.357880] pc : try_oops_init+0x74/0xff8 [oops_tryv1]
[ 1825.363100] lr : try_oops_init+0x64/0xff8 [oops_tryv1]
[ 1825.368311] sp : ffffffc0817f3a50
[ 1825.371663] x29: ffffffc0817f3a50 x28: ffffffd96359ea50 x27: 0000000000000000
[ 1825.378901] x26: ffffffc0817f3bb0 x25: ffffffd96382ac38 x24: ffffffd963633c70
[ 1825.386139] x23: 0000000000000000 x22: 0000000000000000 x21: ffffffd92ac14000
[ 1825.393376] x20: ffffffd92ac6f0d8 x19: ffffffd92ac6f020 x18: 0000000000000006
[ 1825.400612] x17: 3030303030303030 x16: 7830207373657264 x15: 6461204c4c554e20
[ 1825.407848] x14: 656874206f742067 x13: 3030303030303030 x12: 3030303030303030
[ 1825.415086] x11: 00000000000000b7 x10: 0000000000001a30 x9 : ffffffd961e1372c
[ 1825.422322] x8 : ffffff8045b63950 x7 : 0000000000000017 x6 : 0000000000000001
[ 1825.429559] x5 : 0000000000000000 x4 : 0000000000000000 x3 : 0000000000000000
[ 1825.436794] x2 : 0000000000000000 x1 : 0000000000000078 x0 : 0000000000000000
[ 1825.444031] Call trace:
[ 1825.446504]  try_oops_init+0x74/0xff8 [oops_tryv1]
[ 1825.451363]  do_one_initcall+0x60/0x2c0
```

There's two types of information embedded in the Oops diagnostic:
- arch-specific / hardware info : processor & MMU registers, flags, etc
- arch-independent / software info : modules in memory, location of crash (of course it's via the PC/LR or equivalent eegisters), + - very important – the call stack.

```
$ sudo dmesg
[ 1825.203166] oops_tryv1:try_oops_init():37: Lets Oops!
```

```
               Now attempting to write something to the NULL address
0x0000000000000000

<< 1. the arch-specific / hardware info : processor & MMU registers, flags, etc,
follows: >>

[ 1825.216268] Unable to handle kernel NULL pointer dereference at virtual
address 0000000000000000
[ 1825.225198] Mem abort info:
[ 1825.228024]   ESR = 0x0000000096000046
[ 1825.231830]   EC = 0x25: DABT (current EL), IL = 32 bits
[ 1825.237218]   SET = 0, FnV = 0
[ 1825.240356]   EA = 0, S1PTW = 0
[ 1825.243567]   FSC = 0x06: level 2 translation fault
[ 1825.248533] Data abort info:
[ 1825.251445]   ISV = 0, ISS = 0x00000046, ISS2 = 0x00000000
[ 1825.257056]   CM = 0, WnR = 1, TnD = 0, TagAccess = 0
[ 1825.262182]   GCS = 0, Overlay = 0, DirtyBit = 0, Xs = 0
[ 1825.267569] user pgtable: 4k pages, 39-bit VAs, pgdp=00000000481a1000
[ 1825.274103] [0000000000000000] pgd=0800000045835003, p4d=0800000045835003,
pud=0800000045835003, pmd=0000000000000000


<<
Refer the TRM:
ARM ® Architecture Reference Manual
ARMv8, for ARMv8-A architecture profile
(PDF)


Unable to handle kernel NULL pointer dereference at virtual address
0000000000000000

The faulting (virtual) address is held in FAR_Elx:
```

Note: on AArch64 (ARM64), the equivalent register is the **FAR_ELx** that holds the faulting virtual address at Exception Level (EL) n ; n = 1, 2, 3. (FYI, EL0 is userspace, EL1 typically the kernel-space, EL2 the hypervisor, if any, and EL3 the Secure Monitor mode).

From the ARMv8A TRM:

“D12.2.39

FAR_EL1, Fault Address Register (EL1)

The FAR_EL1 characteristics are:

Purpose
Holds the faulting Virtual Address for all synchronous Instruction or Data Abort, PC alignment fault and Watchpoint exceptions that are taken to EL1. ...”

…

```
Mem abort info:
[ 1825.228024]   ESR = 0x0000000096000046
[ 1825.231830]   EC = 0x25: DABT (current EL), IL = 32 bits
[ 1825.237218]   SET = 0, FnV = 0
[ 1825.240356]   EA = 0, S1PTW = 0
...
```

ESR:
D12.2.36 (pg 2759)
ESR_EL1, Exception Syndrome Register (EL1)
…

D2.8.3 Exception syndrome information
…

D1.8.1 (pg 2152)
ESR: “A PC misalignment sets the EC field in the Exception Syndrome Register (ESR) to 0x22 , for the ESR associated with the target Exception level.”

(Interestingly, the ESR value is shown as the Oops bitmask!_):
…

Internal error: Oops: 0000000096000046 [#1] PREEMPT SMP

EC = Exception Class
pg D12-2796

Our EC = 0x25 (i.e. 0010 0101); see it here!
“EC == 0b100101
Data Abort taken without a change in Exception level.
Used for MMU faults generated by data accesses, alignment faults other than those caused by Stack Pointer misalignment, and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug related exceptions.
See ISS encoding for an exception from a Data Abort.”

-a good clue…

DABT = Data Abort !

IL = 32 bits ; IL is Instruction Length

ISV = Instruction Syndrome Valid
“… Indicates whether the syndrome information in ISS[23:0] is valid.
…
ISV is 0 for all faults reported in ESR_EL1 or ESR_EL3. …”

S1PTW
“… S1PTW, bit [7]
For a stage 2 fault, indicates whether the fault was a stage 2 fault on an access made for a stage 1 translation table walk:
0b0 Fault not on a stage 2 translation for a stage 1 translation table walk.
0b1 Fault on the stage 2 translation of an access for a stage 1 translation table walk.
For any abort other than a stage 2 fault this bit is RES0. …”

>>

[1825.284882] Internal error: Oops: 0000000096000046 [#1] PREEMPT SMP
[1825.291236] Modules linked in: oops_tryv1(O+) brcmfmac_wcc rpivid_hevc(C) bcm2835_codec(C) bcm2835_v4l2(C) bcm2835_isp(C) v4l2_mem2mem hci_uart bcm2835_mmal_vchiq(C) btbcm videobuf2_vmalloc videobuf2_dma_contig v3d videobuf2_memops gpu_sched brcmfmac bluetooth videobuf2_v4l2 videodev

```
drm_shmem_helper brcmutil videobuf2_common ecdh_generic ecc raspberrypi_hwmon mc
i2c_brcmstb vc_sm_cma(C) snd_bcm2835(C) raspberrypi_gpiomem uio_pdrv_genirq uio
sch_fq_codel fuse nfnetlink ipv6 [last unloaded: printk_loglvl(O)]

[ 1825.336885] CPU: 3 PID: 4919 Comm: insmod Tainted: G         C O
6.6.22-kenix #1
<<
Context info

Note the 'Tainted flags':
$ cat /proc/sys/kernel/tainted
5248

On host:
$ ~/6.1.25/tools/debugging/kernel-chktaint 5248 2>/dev/null
Kernel is "tainted" for the following reasons:
 * kernel died recently, i.e. there was an OOPS or BUG (#7)
 * staging driver was loaded (#10)
 * externally-built ('out-of-tree') module was loaded  (#12)
For a more detailed explanation of the various taint flags see
 Documentation/admin-guide/tainted-kernels.rst in the Linux kernel sources
 or https://kernel.org/doc/html/latest/admin-guide/tainted-kernels.html
Raw taint value as int/string: 5248/'G      D  C O      '


6.6.22-kenix: kernel version (build #1)
>>

[ 1825.344913] Hardware name: Raspberry Pi 4 Model B Rev 1.4 (DT)
[ 1825.350823] pstate: 60000005 (nZCv daif -PAN -UAO -TCO -DIT -SSBS BTYPE=--)

<<
From the ref manual:

pg 2150:
```

D1.7.1 Accessing PSTATE fields

In AArch64 state, PSTATE fields can be accessed using Special-purpose registers that can be directly read using the MRS instruction, and directly written using the MSR (register) instructions. Table D1-5 shows the Special-purpose registers that access the PSTATE fields that hold AArch64 state, when the PE is in AArch64 state. All other PSTATE fields do not have direct read and write access.

Table D1-5 Accessing PSTATE fields using MRS and MSR (register)

Special-purpose register	PSTATE fields
NZCV	N, Z, C, V
DAIF	D, A, I, F
CurrentEL	EL
SPSel	SP
PAN	PAN
UAO	UAO
DIT	DIT

```
NZCV condition flags:
https://developer.arm.com/documentation/ddi0601/2024-03/AArch64-Registers/NZCV--Condition-Flags
nZCv => n,v clear & Z,C were set
```

```
DAIF: interrupt mask bits
daif => all were clear

CurrentEL – current Exception Level (will be 1)

SPSel – SP selected

PAN: Privileged Access Never
('-PAN' in the Oops implies it isn't set)

UAO - User Access Override (UAO) bit.
-UAO

DIT - Data Independent Timing (DIT) bit.
-DIT
>>

<< 2. the arch-independent / software info : modules in memory, location of
crash (of course it's via the PC/LR or equivalent registers), + - very important
– the call stack, follows:
>>

[ 1825.357880] pc : try_oops_init+0x74/0xff8 [oops_tryv1]
[ 1825.363100] lr : try_oops_init+0x64/0xff8 [oops_tryv1]
[ 1825.368311] sp : ffffffc0817f3a50
[ 1825.371663] x29: ffffffc0817f3a50 x28: ffffffd96359ea50 x27: 0000000000000000
[ 1825.378901] x26: ffffffc0817f3bb0 x25: ffffffd96382ac38 x24: ffffffd963633c70
[ 1825.386139] x23: 0000000000000000 x22: 0000000000000000 x21: ffffffd92ac14000
[ 1825.393376] x20: ffffffd92ac6f0d8 x19: ffffffd92ac6f020 x18: 0000000000000006
[ 1825.400612] x17: 3030303030303030 x16: 7830207373657264 x15: 6461204c4c554e20
[ 1825.407848] x14: 656874206f742067 x13: 3030303030303030 x12: 3030303030303030
[ 1825.415086] x11: 00000000000000b7 x10: 0000000000001a30 x9 : ffffffd961e1372c
[ 1825.422322] x8 : ffffff8045b63950 x7 : 0000000000000017 x6 : 0000000000000001
[ 1825.429559] x5 : 0000000000000000 x4 : 0000000000000000 x3 : 0000000000000000
[ 1825.436794] x2 : 0000000000000000 x1 : 0000000000000078 x0 : 0000000000000000
[ 1825.444031] Call trace:
[ 1825.446504]  try_oops_init+0x74/0xff8 [oops_tryv1]
[ 1825.451363]  do_one_initcall+0x60/0x2c0
[ 1825.455252]  do_init_module+0x60/0x210
[ 1825.459053]  load_module+0x1f4c/0x2038
[ 1825.462851]  init_module_from_file+0x90/0xe0
[ 1825.467177]  __arm64_sys_finit_module+0x1e4/0x2f8
[ 1825.471944]  invoke_syscall+0x50/0x120
[ 1825.475742]  el0_svc_common.constprop.0+0x48/0xf0
[ 1825.480509]  do_el0_svc+0x24/0x38
[ 1825.483865]  el0_svc+0x40/0xe8
[ 1825.486958]  el0t_64_sync_handler+0x120/0x130
[ 1825.491371]  el0t_64_sync+0x190/0x198
[ 1825.495081] Code: 395402a0 370001e0 d2800000 52800f01 (b9000001)
[ 1825.501256] ---[ end trace 0000000000000000 ]---
```

Analyzing / Debugging this Oops

First off, you need some debug info embedded inside the .ko file… if kernel config is NO debug info, these approaches won't work…

So, I rebuilt the (Yocto) system kernel with a few kernel debug configs enabled (certainly not all are required – f.e. lock debug, KASAN, KCSAN, etc, right now):

```
CONFIG_DEBUG_INFO=y
CONFIG_DEBUG_INFO_DWARF5=y
CONFIG_DEBUG_INFO_COMPRESSED_NONE=y
CONFIG_GDB_SCRIPTS=y
CONFIG_LOCKUP_DETECTOR=y
CONFIG_SOFTLOCKUP_DETECTOR=y
CONFIG_HARDLOCKUP_DETECTOR=y
CONFIG_HARDLOCKUP_DETECTOR_BUDDY=y
CONFIG_HARDLOCKUP_DETECTOR_COUNTS_HRTIMER=y
CONFIG_WQ_WATCHDOG=y
CONFIG_WQ_CPU_INTENSIVE_REPORT=y
CONFIG_DEBUG_ATOMIC_SLEEP=y
```

built it on the target (oops_simple.ko size is now ~ 260 KB compared to just 80 KB without debug info):

```
$ file ./oops_simple.ko
./oops_simple.ko: ELF 64-bit LSB relocatable, ARM aarch64, version 1 (SYSV),
BuildID[sha1]=627d..., with debug_info, not stripped
```

and then ran it on the R Pi 4B:

```
[  780.087012] Hello, about to Oops!
[  780.090788] Unable to handle kernel NULL pointer dereference at virtual
address 0000000000000040
[  780.099774] Mem abort info:
[  780.102642]   ESR = 0x0000000096000046
[  780.106456]   EC = 0x25: DABT (current EL), IL = 32 bits
[  780.111873]   SET = 0, FnV = 0
[  780.115002]   EA = 0, S1PTW = 0
[  780.118189]   FSC = 0x06: level 2 translation fault
[  780.123136] Data abort info:
[  780.126056]   ISV = 0, ISS = 0x00000046, ISS2 = 0x00000000
[  780.131620]   CM = 0, WnR = 1, TnD = 0, TagAccess = 0
[  780.136743]   GCS = 0, Overlay = 0, DirtyBit = 0, Xs = 0
[  780.142132] user pgtable: 4k pages, 39-bit VAs, pgdp=0000000041144000
[  780.148665] [0000000000000040] pgd=0800000040ee3003, p4d=0800000040ee3003,
pud=0800000040ee3003, pmd=0000000000000000
[  780.159442] Internal error: Oops: 0000000096000046 [#1] PREEMPT SMP
[  780.165794] Modules linked in: oops_simple(O+) hci_uart btbcm brcmfmac_wcc
bluetooth rpivid_hevc(C) bcm2835_v4l2(C) bcm2835_codec(C) bcm2835_isp(C)
videobuf2_vmalloc bcm2
835_mmal_vchiq(C) v4l2_mem2mem brcmfmac videobuf2_dma_contig videobuf2_memops
videobuf2_v4l2 v3d videodev gpu_sched drm_shmem_helper videobuf2_common
ecdh_generic brcmutil r
aspberrypi_hwmon mc snd_bcm2835(C) ecc i2c_brcmstb vc_sm_cma(C)
raspberrypi_gpiomem uio_pdrv_genirq uio sch_fq_codel fuse nfnetlink ipv6
[  780.208531] CPU: 2 PID: 3745 Comm: insmod Tainted: G         C O
6.6.22-kenix #1
[  780.216559] Hardware name: Raspberry Pi 4 Model B Rev 1.4 (DT)
[  780.222468] pstate: 60000005 (nZCv daif -PAN -UAO -TCO -DIT -SSBS BTYPE=--)
[  780.229526] pc : oops2_init+0x30/0xff8 [oops_simple]
[  780.234566] lr : oops2_init+0x20/0xff8 [oops_simple]
[  780.239602] sp : ffffffc08271ba70
[  780.242954] x29: ffffffc08271ba70 x28: ffffffdaaadaea50 x27: 0000000000000000
[  780.250192] x26: ffffffc08271bbb0 x25: ffffffdaab03c070 x24: ffffffdaaae43d30
[  780.257430] x23: 0000000000000000 x22: 0000000000000000 x21: 0000000000000000
[  780.264667] x20: ffffff804d289dc0 x19: ffffffda491e9008 x18: 0000000000000000
[  780.271903] x17: 0000000000000000 x16: 0000000000000000 x15: 0000000000000000
[  780.279139] x14: 0000b62fb918d452 x13: 01abdb323e3cec8c x12: 00000000fa83b2da
[  780.286375] x11: 00000000000003b0 x10: 0000000000001a40 x9 : ffffffdaa961cc8c
[  780.293612] x8 : ffffff8040803960 x7 : 00000000000000f6 x6 : 000000000000b726
```

```
[  780.300848] x5 : 0000000000000000 x4 : 0000000000000000 x3 : 0000000000000000
[  780.308085] x2 : 000000000000abcd x1 : 0000000000000000 x0 : 0000000000000000
[  780.315321] Call trace:
[  780.317794]  oops2_init+0x30/0xff8 [oops_simple]
[  780.322478]  do_one_initcall+0x60/0x2c0
[  780.326366]  do_init_module+0x60/0x210
[  780.330164]  load_module+0x1f64/0x2050
[  780.333962]  init_module_from_file+0x90/0xe0
[  780.338287]  __arm64_sys_finit_module+0x1e4/0x2f8
[  780.343054]  invoke_syscall+0x50/0x120
[  780.346854]  el0_svc_common.constprop.0+0xc8/0xf0
[  780.351620]  do_el0_svc+0x24/0x38
[  780.354977]  el0_svc+0x40/0xe8
[  780.358070]  el0t_64_sync_handler+0x120/0x130
[  780.362484]  el0t_64_sync+0x190/0x198
[  780.366194] Code: 90ffffe1 d29579a2 52800000 f9428021 (f9002022)
[  780.372367] ---[ end trace 0000000000000000 ]---
```

Most important:
pc : oops2_init+0x30/0xff8 [oops_simple]

1. Kernel debug info available: let's use objdump

$ objdump -dS ./oops_simple.ko

```
./oops_simple.ko:     file format elf64-littleaarch64


Disassembly of section .text:

0000000000000000 <delay_sec-0x8>:
   0:   d503201f        nop

--snip--

<< The Oops occurred in the function oops2_init() at a start offset of 0x30 ;
so look for that…
>>

static int __init oops2_init(void)
{
   8:   d503201f        nop
   c:   d503201f        nop
  10:   d503233f        paciasp
  14:   a9bf7bfd        stp     x29, x30, [sp, #-16]!
        if (!f1)
                return -ENOMEM;
        pr_info("sizeof(long) = %ld, sizeof(struct faker) = %lu, actual space
alloced = %lu\n",
            sizeof(long), sizeof(struct faker), ksize(f1));
#else
        pr_info("Hello, about to Oops!\n");
  18:   90000000        adrp    x0, 0 <init_module-0x8>
{
  1c:   910003fd        mov     x29, sp
        pr_info("Hello, about to Oops!\n");
  20:   91000000        add     x0, x0, #0x0
  24:   94000000        bl      0 <_printk>
#endif
        f1->bad_cache_align = 0xabcd;
```

Offset column

```
  28:   90000001        adrp    x1, 0 <init_module-0x8>
  2c:   d29579a2        mov     x2, #0xabcd                     // #43981
        return 0;
}
  30:   52800000        mov     w0, #0x0                        // #0
        f1->bad_cache_align = 0xabcd;
  34:   f9400021        ldr     x1, [x1]

--snip--
```

Got it!

2. Next approach: via GDB !

```
$ gdb -q ./oops_simple.ko
Reading symbols from ./oops_simple.ko...

warning: could not convert 'main' from the host encoding (ANSI_X3.4-1968) to
UTF-32.
This normally should not happen, please file a bug report.
(gdb) list *oops2_init+0x30
0x38 is in oops2_init
(/home/.../k_oops_warn_panic/oops_simple/oops_simple.c:32).
27              pr_info("sizeof(long) = %ld, sizeof(struct faker) = %lu, actual
space alloced = %lu\n",
28                      sizeof(long), sizeof(struct faker), ksize(f1));
29      #else
30              pr_info("Hello, about to Oops!\n");
31      #endif
32              f1->bad_cache_align = 0xabcd;
33              return 0;
34      }
```

Perfect!

3. Next approach: via addr2line !

```
$ addr2line -e ./oops_simple.ko oops2_init+0x30
/home/yo/kaiwanTECH/L5_kernel_debug/k_oops_warn_panic/oops_simple/../../
convenient.h:282 (discriminator 1)
```

Fails..

4. Next approach: via faddr2line !

Use when:
- addr2line fails
- KASLR is enabled (typically the case).

```
$ /usr/src/kernel/scripts/faddr2line ./oops_simple.ko oops2_init+0x30/0xff8
skipping oops2_init address at 0x38 due to size mismatch (0xff8 != 0x40)
no match for oops2_init+0x30/0xff8
kenix-raspberrypi4-64-dca632f7ca5a oops_simple $
```

Oops, it failed…
Try just giving the offset..

```
$ /usr/src/kernel/scripts/faddr2line ./oops_simple.ko oops2_init+0x30
oops2_init+0x30/0x40:
oops2_init at /home/.../k_oops_warn_panic/oops_simple/oops_simple.c:32
```

Perfect.

5. Next approach: via the decodecode script:

```
$ /usr/src/kernel/scripts/decodecode < oops1
[ 780.366194] Code: 90ffffe1 d29579a2 52800000 f9428021 (f9002022)
All code
========
   0:   90ffffe1        adrp    x1, 0xfffffffffffc000
   4:   d29579a2        mov     x2, #0xabcd                     // #43981
   8:   52800000        mov     w0, #0x0                        // #0
   c:   f9428021        ldr     x1, [x1, #1280]
  10:*  f9002022        str     x2, [x1, #64]           <-- trapping instruction

Code starting with the faulting instruction
===========================================
   0:   f9002022        str     x2, [x1, #64]
$
```

This is the assembler following the buggy C code line! `(see via objdump -dS).`

6. Next approach: via the decode_stacktrace.sh script:

Doesn't work here (right now); as we require the uncompressed kernel image with debug symbols - the *vmlinux* image – to try it…

So: on the build host:

Look for the vmlinux image file:
first, get the working dir where the kernel was built:

```
$ bitbake -e linux-raspberrypi |grep "^WORKDIR="
WORKDIR="/big.../tmp-glibc/work/raspberrypi4_64-kaiwanTECH-linux/linux-
raspberrypi/6.6.22+git"
$
```

Search there:

```
build $ find
/big/.../build/tmp-glibc/work/raspberrypi4_64-kaiwanTECH-linux/linux-
raspberrypi/6.6.22+git -name "vmlinux*"
/big/.../build/tmp-glibc/work/raspberrypi4_64-kaiwanTECH-linux/linux-
raspberrypi/6.6.22+git/package/boot/vmlinux-6.6.22-kenix
/big/.../build/tmp-glibc/work/raspberrypi4_64-kaiwanTECH-linux/linux-
raspberrypi/6.6.22+git/package/boot/.debug/vmlinux-6.6.22-kenix
…

$ file /big/.../build/tmp-glibc/work/raspberrypi4_64-kaiwanTECH-linux/linux-
raspberrypi/6.6.22+git/package/boot/.debug/vmlinux-6.6.22-kenix
/big/.../build/tmp-glibc/work/raspberrypi4_64-kaiwanTECH-linux/linux-
raspberrypi/6.6.22+git/package/boot/.debug/vmlinux-6.6.22-kenix: ELF 64-bit LSB
```

```
pie executable, ARM aarch64, version 1 (SYSV), statically linked,
BuildID[sha1]=3c0b0..., with debug_info, not stripped
```

Copy (scp) it across to the target for kernel debug purposes! (it can be large; close to 300 MB here)..

FYI, an example Oops from within an interrupt (irq) context:

From the LKD book repo:

```
# cd ch7/oops_inirqv3
# make
...
# insmod oops_inirqv3.ko
[19676.210163] oops_inirqv3: loading out-of-tree module taints kernel.
[19676.217590] Unable to handle kernel NULL pointer dereference at virtual
address 0000000000000100
[19676.226508] Mem abort info:
[19676.229334]   ESR = 0x0000000096000046
[19676.233130]   EC = 0x25: DABT (current EL), IL = 32 bits
[19676.238514]   SET = 0, FnV = 0
[19676.241604]   EA = 0, S1PTW = 0
[19676.244783]   FSC = 0x06: level 2 translation fault
[19676.249724] Data abort info:
[19676.252637]   ISV = 0, ISS = 0x00000046, ISS2 = 0x00000000
[19676.258194]   CM = 0, WnR = 1, TnD = 0, TagAccess = 0
[19676.263311]   GCS = 0, Overlay = 0, DirtyBit = 0, Xs = 0
[19676.268694] user pgtable: 4k pages, 39-bit VAs, pgdp=0000000044666000
[19676.275222] [0000000000000100] pgd=08000000447e0003, p4d=08000000447e0003,
pud=08000000447e0003, pmd=0000000000000000
[19676.285993] Internal error: Oops: 0000000096000046 [#1] PREEMPT SMP
[19676.292347] Modules linked in: oops_inirqv3(O+) hci_uart btbcm bluetooth
brcmfmac_wcc rpivid_hevc(C) bcm2835_codec(C) bcm2835_v4l2(C) bcm2835_isp(C)
v4l2_me
m2mem bcm2835_mmal_vchiq(C) videobuf2_vmalloc videobuf2_dma_contig
videobuf2_memops videobuf2_v4l2 brcmfmac videodev v3d gpu_sched drm_shmem_helper
brcmutil vi
deobuf2_common ecdh_generic ecc i2c_brcmstb raspberrypi_hwmon mc snd_bcm2835(C)
vc_sm_cma(C) raspberrypi_gpiomem uio_pdrv_genirq uio sch_fq_codel fuse nfnetlin
k ipv6
[19676.335189] CPU: 1 PID: 50016 Comm: insmod Tainted: G         C O
6.6.22-kenix #1
[19676.343307] Hardware name: Raspberry Pi 4 Model B Rev 1.4 (DT)
[19676.349218] pstate: 000000c5 (nzcv daIF -PAN -UAO -TCO -DIT -SSBS BTYPE=--)
[19676.356278] pc : irq_work+0x3c/0x70 [oops_inirqv3]
[19676.361146] lr : irq_work_single+0x30/0x88
[19676.365300] sp : ffffffc08000beb0
[19676.368654] x29: ffffffc08000beb0 x28: ffffff8045ac3d80 x27: 0000000000000000
[19676.375895] x26: ffffffc083bd3bb0 x25: ffffffd1cb63c070 x24: ffffffd1cb443d30
[19676.383134] x23: 0000000000000001 x22: 0000000000000005 x21: 0000000000000023
[19676.390374] x20: 0000000000000022 x19: ffffffd1576c6500 x18: 0000000000000006
[19676.397613] x17: ffffffaeb49da000 x16: ffffffc080008000 x15: 0000000000000000
[19676.404852] x14: 0000000000000000 x13: 0064692d646c6975 x12: 622e756e672e6574
[19676.412091] x11: 000000000040003b x10: 0000000000012cc0 x9 : ffffffd1c9c1cf50
[19676.419331] x8 : ffffffc083bd3a48 x7 : 0000000000000000 x6 : 000000000000003f
[19676.426571] x5 : ffffffd1c9a100e0 x4 : ffffffaeb49da000 x3 : 0000000000000001
[19676.433810] x2 : 0000000000000000 x1 : 0000000000000078 x0 : 0000000000000100
[19676.441049] Call trace:
[19676.443522]  irq_work+0x3c/0x70 [oops_inirqv3]
```

```
[19676.448033]  irq_work_run_list+0x4c/0x68
[19676.452006]  irq_work_run+0x28/0x48
[19676.455538]  ipi_handler+0x1b8/0x1f8
[19676.459165]  handle_percpu_devid_irq+0x90/0x238
[19676.463758]  generic_handle_domain_irq+0x34/0x58
[19676.468440]  gic_handle_irq+0x4c/0xd8
[19676.472148]  call_on_irq_stack+0x24/0x58
[19676.476123]  do_interrupt_handler+0x88/0x98
[19676.480363]  el1_interrupt+0x34/0x68
[19676.483988]  el1h_64_irq_handler+0x18/0x28
[19676.488140]  el1h_64_irq+0x64/0x68
[19676.491584]  do_one_initcall+0x98/0x2c0
[19676.495469]  do_init_module+0x60/0x210
[19676.499269]  load_module+0x1f64/0x2050
[19676.503067]  init_module_from_file+0x90/0xe0
[19676.507395]  __arm64_sys_finit_module+0x1e4/0x2f8
[19676.512163]  invoke_syscall+0x50/0x120
[19676.515963]  el0_svc_common.constprop.0+0x48/0xf0
[19676.520732]  do_el0_svc+0x24/0x38
[19676.524090]  el0_svc+0x40/0xe8
[19676.527183]  el0t_64_sync_handler+0x120/0x130
[19676.531599]  el0t_64_sync+0x190/0x198
[19676.535311] Code: 72181c1f 540000a0 d2802000 52800f01 (b9000001)
[19676.541487] ---[ end trace 0000000000000000 ]---
[19676.546165] Kernel panic - not syncing: Oops: Fatal exception in interrupt
[19676.553134] SMP: stopping secondary CPUs
[19676.557109] Kernel Offset: 0x1149a00000 from 0xffffffc080000000
[19676.563107] PHYS_OFFSET: 0x0
[19676.566021] CPU features: 0x0,80000201,3c020000,0000421b
[19676.571404] Memory Limit: none
[19676.574496] ---[ end Kernel panic - not syncing: Oops: Fatal exception in
interrupt ]---
```
