ARM64 Kernel Booting Process

This document describes boot loader requirements to boot Kernel, ARM64 Virtual Memory Layout, ARM64 IRQ Vectors Setup, FDT mapping and ARM64 Kernel booting process.

1. boot loader requirements to boot Kernel

Boot loader simply to define all software that executes on the CPU(s) before control is passed to the Linux kernel. This may include secure monitor and hypervisor code, or it may just be a handful of instructions for preparing a minimal boot environment.

Essentially, the boot loader should provide the following:

(1) Setup and initialize the RAM
(2) Setup the device tree
(3) Decompress the kernel image
(4) Call the kernel image

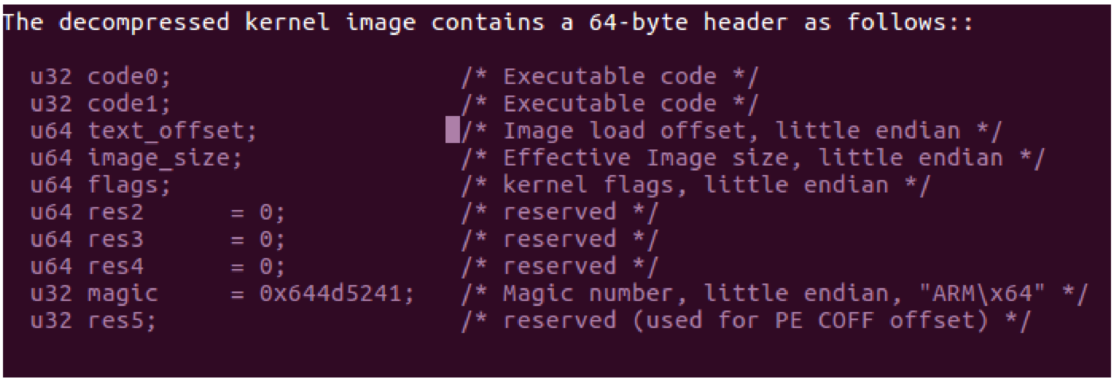
Setup and initialize the RAM.
Requirement: MANDATORY
The boot loader is expected to find and initialize all RAM that the kernel will use for volatile data storage in the system. It performs this in a machine dependent manner. (It may use internal algorithms to automatically locate and size all RAM, or it may use knowledge of the RAM in the machine, or any other method the boot loader designer sees fit.)

Setup the device tree
Requirement: MANDATORY
The device tree blob (dtb) must be placed on an 8-byte boundary and must not exceed 2 megabytes in size. Since the dtb will be mapped cacheable using blocks of up to 2 megabytes in size, it must not be placed within any 2M region which must be mapped with any specific attributes.

Decompress the kernel image
Requirement: OPTIONAL
The AArch64 kernel does not currently provide a decompressor and therefore requires decompression (gzip etc.) to be performed by the boot loader if a compressed Image target (e.g. Image.gz) is used. For bootloaders that do not implement this requirement, the uncompressed Image target is available instead.

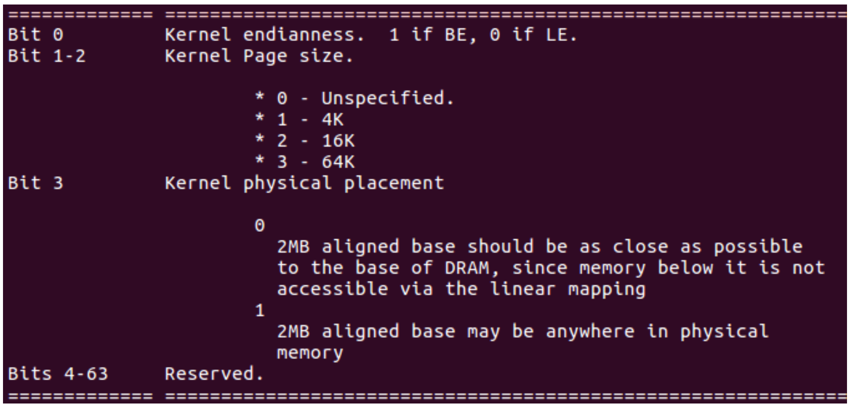
Call the kernel image
Requirement: MANDATORY

Kernel Image Header



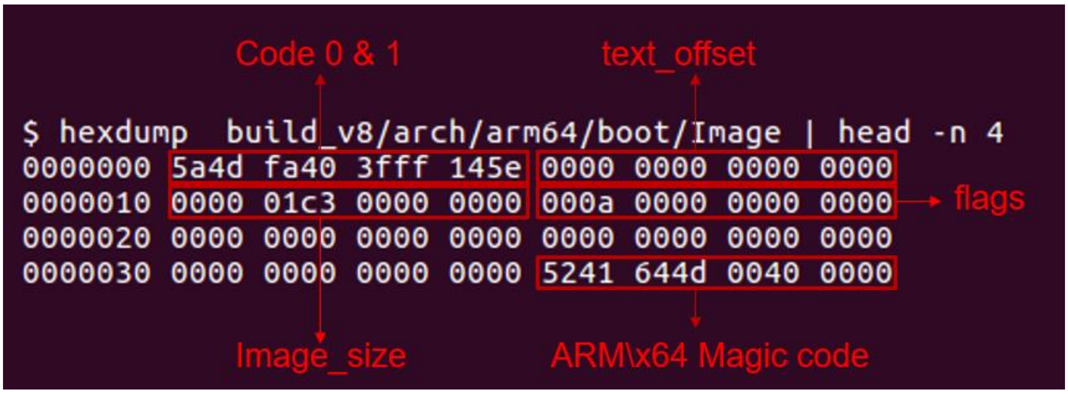
```
The decompressed kernel image contains a 64-byte header as follows::

  u32 code0;                    /* Executable code */
  u32 code1;                    /* Executable code */
  u64 text_offset;              /* Image load offset, little endian */
  u64 image_size;               /* Effective Image size, little endian */
  u64 flags;                    /* kernel flags, little endian */
  u64 res2      = 0;            /* reserved */
  u64 res3      = 0;            /* reserved */
  u64 res4      = 0;            /* reserved */
  u32 magic     = 0x644d5241;   /* Magic number, little endian, "ARM\x64" */
  u32 res5;                     /* reserved (used for PE COFF offset) */
```

Kernel Image Header: Flags



```
============= =============================================================
Bit 0         Kernel endianness.  1 if BE, 0 if LE.
Bit 1-2       Kernel Page size.

                * 0 - Unspecified.
                * 1 - 4K
                * 2 - 16K
                * 3 - 64K
Bit 3         Kernel physical placement

                0
                  2MB aligned base should be as close as possible
                  to the base of DRAM, since memory below it is not
                  accessible via the linear mapping
                1
                  2MB aligned base may be anywhere in physical
                  memory
Bits 4-63     Reserved.
============= =============================================================
```

Example: If 5.10.y Kernel Image



```
$ hexdump  build_v8/arch/arm64/boot/Image | head -n 4
0000000 5a4d fa40 3fff 145e 0000 0000 0000 0000
0000010 0000 01c3 0000 0000 000a 0000 0000 0000
0000020 0000 0000 0000 0000 0000 0000 0000 0000
0000030 0000 0000 0000 0000 5241 644d 0040 0000
```

Other constrains before jumping into kernel

- Quiesce all DMA
- Primary CPU register settings

  x0 = dt blob address in RAM

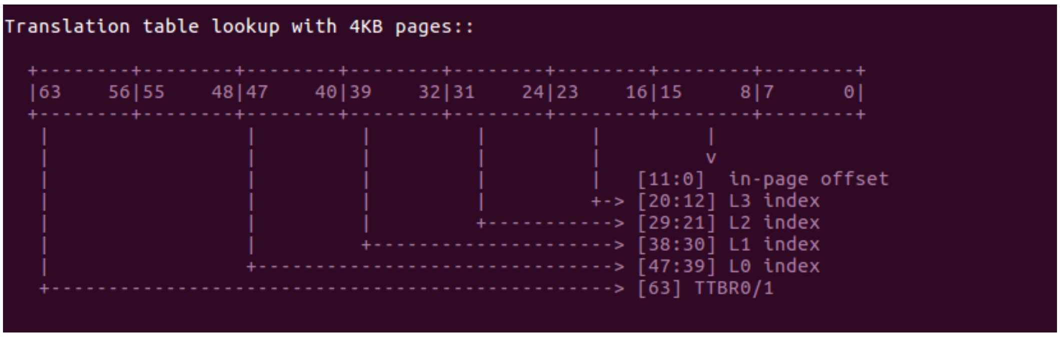
  x1-x3 = 0
- MMU off, Instruction cache either on or off

  … …

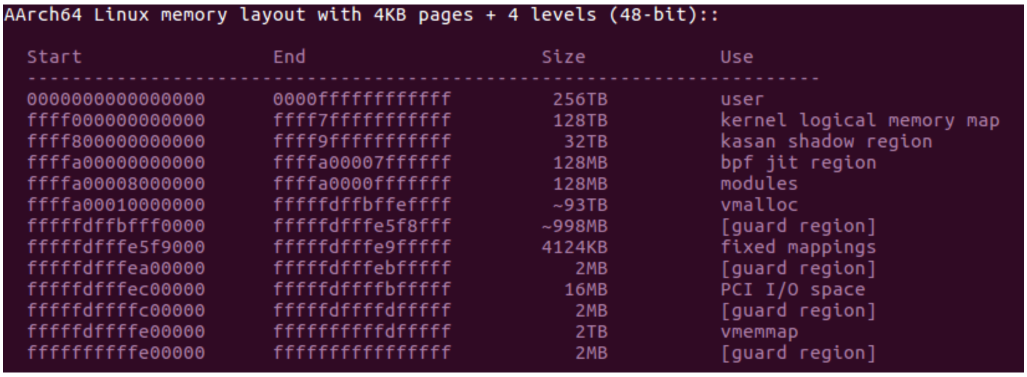
Please refer to Documentation/arm64/booting.rst.

2. ARM64 Virtual Memory Layout

AArch64 Linux memory layout with 4KB pages + 4 levels (48-bit):



```
Translation table lookup with 4KB pages::

  +--------+--------+--------+--------+--------+--------+--------+--------+
  |63    56|55    48|47    40|39    32|31    24|23    16|15     8|7      0|
  +--------+--------+--------+--------+--------+--------+--------+--------+
   |                 |         |         |         |         |
   |                 |         |         |         |         v
   |                 |         |         |         |   [11:0]  in-page offset
   |                 |         |         |         +-> [20:12] L3 index
   |                 |         |         +-----------> [29:21] L2 index
   |                 |         +---------------------> [38:30] L1 index
   |                 +-------------------------------> [47:39] L0 index
   +-------------------------------------------------> [63] TTBR0/1
```

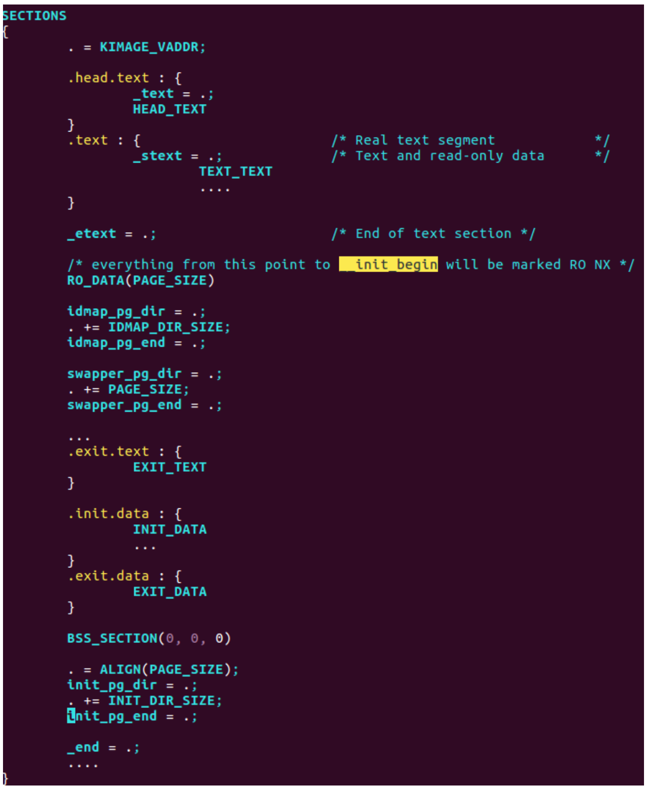


```
AArch64 Linux memory layout with 4KB pages + 4 levels (48-bit)::

  Start                 End                      Size           Use
  -----------------------------------------------------------------------
  0000000000000000      0000ffffffffffff          256TB         user
  ffff000000000000      ffff7fffffffffff          128TB         kernel logical memory map
  ffff800000000000      ffff9fffffffffff           32TB         kasan shadow region
  ffffa00000000000      ffffa00007ffffff          128MB         bpf jit region
  ffffa00008000000      ffffa0000fffffff          128MB         modules
  ffffa00010000000      fffffdffbffeffff          ~93TB         vmalloc
  fffffdffbfff0000      fffffdfffe5f8fff         ~998MB         [guard region]
  fffffdfffe5f9000      fffffdfffe9fffff         4124KB         fixed mappings
  fffffdfffea00000      fffffdfffebfffff            2MB         [guard region]
  fffffdfffec00000      fffffdffffbfffff           16MB         PCI I/O space
  fffffdffffc00000      fffffdffffdfffff            2MB         [guard region]
  fffffdffffe00000      ffffffffffdfffff            2TB         vmemmap
  ffffffffffe00000      ffffffffffffffff            2MB         [guard region]
```

arch/arm64/kernel/vmlinux.lds.S

Note: kernel image address is started from vmalloc space

#define KIMAGE_VADDR   (MODULES_END)



```
SECTIONS
{
        . = KIMAGE_VADDR;

        .head.text : {
                _text = .;
                HEAD_TEXT
        }
        .text : {                       /* Real text segment            */
                _stext = .;             /* Text and read-only data      */
                        TEXT_TEXT
                        ....
        }

        _etext = .;                     /* End of text section */

        /* everything from this point to __init_begin will be marked RO NX */
        RO_DATA(PAGE_SIZE)

        idmap_pg_dir = .;
        . += IDMAP_DIR_SIZE;
        idmap_pg_end = .;

        swapper_pg_dir = .;
        . += PAGE_SIZE;
        swapper_pg_end = .;

        ...
        .exit.text : {
                EXIT_TEXT
        }

        .init.data : {
                INIT_DATA
                ...
        }
        .exit.data : {
                EXIT_DATA
        }

        BSS_SECTION(0, 0, 0)

        . = ALIGN(PAGE_SIZE);
        init_pg_dir = .;
        . += INIT_DIR_SIZE;
        init_pg_end = .;

        _end = .;
        ....
}
```

3. ARM64 IRQ Vectors Setup

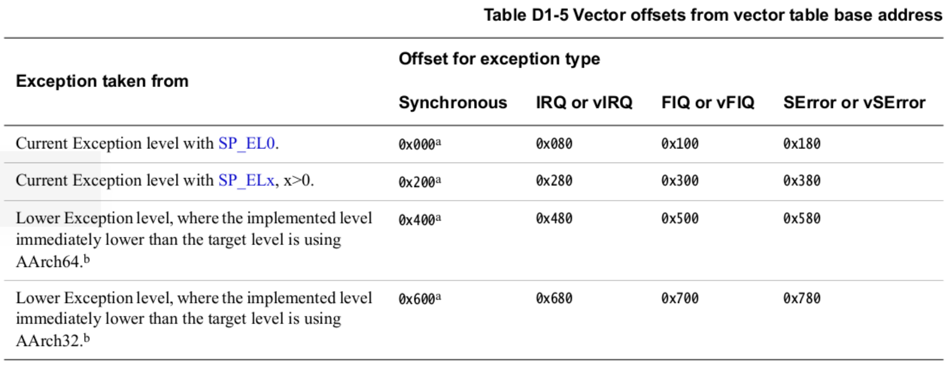
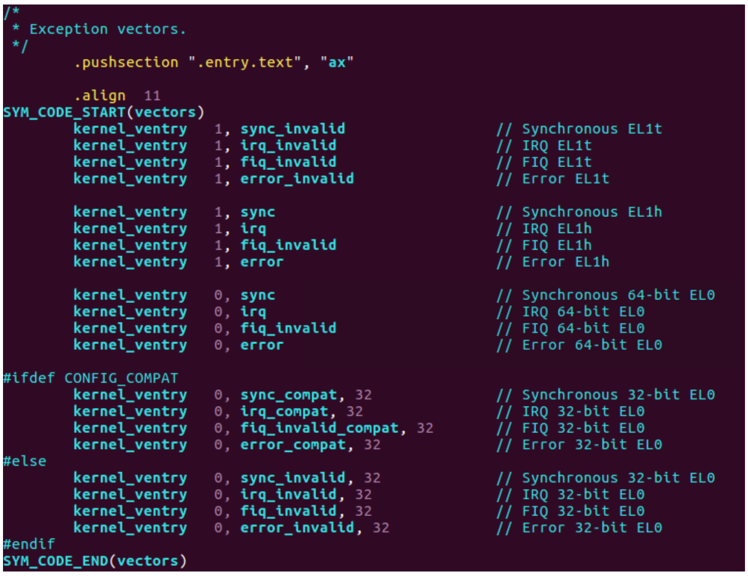


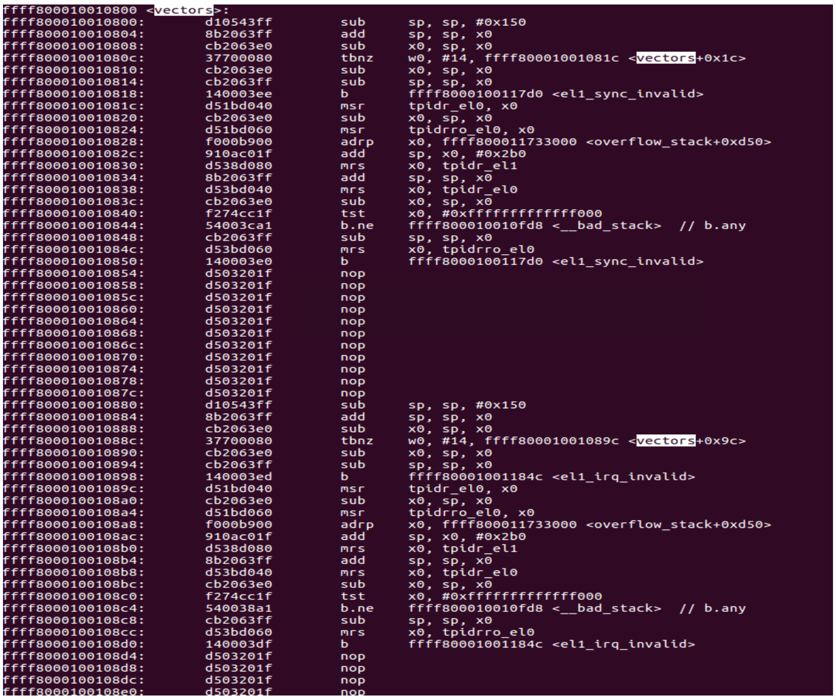
Table D1-5 Vector offsets from vector table base address

Exception taken from	Offset for exception type			
	Synchronous	IRQ or vIRQ	FIQ or vFIQ	SError or vSError
Current Exception level with SP_EL0.	0x000[a]	0x080	0x100	0x180
Current Exception level with SP_ELx, x>0.	0x200[a]	0x280	0x300	0x380
Lower Exception level, where the implemented level immediately lower than the target level is using AArch64.[b]	0x400[a]	0x480	0x500	0x580
Lower Exception level, where the implemented level immediately lower than the target level is using AArch32.[b]	0x600[a]	0x680	0x700	0x780



```
/*
 * Exception vectors.
 */
        .pushsection ".entry.text", "ax"

        .align  11
SYM_CODE_START(vectors)
        kernel_ventry   1, sync_invalid                 // Synchronous EL1t
        kernel_ventry   1, irq_invalid                  // IRQ EL1t
        kernel_ventry   1, fiq_invalid                  // FIQ EL1t
        kernel_ventry   1, error_invalid                // Error EL1t

        kernel_ventry   1, sync                         // Synchronous EL1h
        kernel_ventry   1, irq                          // IRQ EL1h
        kernel_ventry   1, fiq_invalid                  // FIQ EL1h
        kernel_ventry   1, error                        // Error EL1h

        kernel_ventry   0, sync                         // Synchronous 64-bit EL0
        kernel_ventry   0, irq                          // IRQ 64-bit EL0
        kernel_ventry   0, fiq_invalid                  // FIQ 64-bit EL0
        kernel_ventry   0, error                        // Error 64-bit EL0

#ifdef CONFIG_COMPAT
        kernel_ventry   0, sync_compat, 32              // Synchronous 32-bit EL0
        kernel_ventry   0, irq_compat, 32               // IRQ 32-bit EL0
        kernel_ventry   0, fiq_invalid_compat, 32       // FIQ 32-bit EL0
        kernel_ventry   0, error_compat, 32             // Error 32-bit EL0
#else
        kernel_ventry   0, sync_invalid, 32             // Synchronous 32-bit EL0
        kernel_ventry   0, irq_invalid, 32              // IRQ 32-bit EL0
        kernel_ventry   0, fiq_invalid, 32              // FIQ 32-bit EL0
        kernel_ventry   0, error_invalid, 32            // Error 32-bit EL0
#endif
SYM_CODE_END(vectors)
```

Vectors objdump:



```
ffff800010010800 <vectors>:
ffff800010010800:       d10543ff        sub     sp, sp, #0x150
ffff800010010804:       8b2063ff        add     sp, sp, x0
ffff800010010808:       cb2063e0        sub     x0, sp, x0
ffff80001001080c:       37700080        tbnz    w0, #14, ffff80001001081c <vectors+0x1c>
ffff800010010810:       cb2063e0        sub     x0, sp, x0
ffff800010010814:       cb2063ff        sub     sp, sp, x0
ffff800010010818:       140003ee        b       ffff8000100117d0 <el1_sync_invalid>
ffff80001001081c:       d51bd040        msr     tpidr_el0, x0
ffff800010010820:       cb2063e0        sub     x0, sp, x0
ffff800010010824:       d51bd060        msr     tpidrro_el0, x0
ffff800010010828:       f000b900        adrp    x0, ffff800011733000 <overflow_stack+0xd50>
ffff80001001082c:       910ac01f        add     sp, x0, #0x2b0
ffff800010010830:       d538d080        mrs     x0, tpidr_el1
ffff800010010834:       8b2063ff        add     sp, sp, x0
ffff800010010838:       d53bd040        mrs     x0, tpidr_el0
ffff80001001083c:       cb2063e0        sub     x0, sp, x0
ffff800010010840:       f274cc1f        tst     x0, #0xfffffffffffff000
ffff800010010844:       54003ca1        b.ne    ffff800010010fd8 <__bad_stack>  // b.any
ffff800010010848:       cb2063ff        sub     sp, sp, x0
ffff80001001084c:       d53bd060        mrs     x0, tpidrro_el0
ffff800010010850:       140003e0        b       ffff8000100117d0 <el1_sync_invalid>
ffff800010010854:       d503201f        nop
ffff800010010858:       d503201f        nop
ffff80001001085c:       d503201f        nop
ffff800010010860:       d503201f        nop
ffff800010010864:       d503201f        nop
ffff800010010868:       d503201f        nop
ffff80001001086c:       d503201f        nop
ffff800010010870:       d503201f        nop
ffff800010010874:       d503201f        nop
ffff800010010878:       d503201f        nop
ffff80001001087c:       d503201f        nop
ffff800010010880:       d10543ff        sub     sp, sp, #0x150
ffff800010010884:       8b2063ff        add     sp, sp, x0
ffff800010010888:       cb2063e0        sub     x0, sp, x0
ffff80001001088c:       37700080        tbnz    w0, #14, ffff80001001089c <vectors+0x9c>
ffff800010010890:       cb2063e0        sub     x0, sp, x0
ffff800010010894:       cb2063ff        sub     sp, sp, x0
ffff800010010898:       140003ed        b       ffff80001001184c <el1_irq_invalid>
ffff80001001089c:       d51bd040        msr     tpidr_el0, x0
ffff8000100108a0:       cb2063e0        sub     x0, sp, x0
ffff8000100108a4:       d51bd060        msr     tpidrro_el0, x0
ffff8000100108a8:       f000b900        adrp    x0, ffff800011733000 <overflow_stack+0xd50>
ffff8000100108ac:       910ac01f        add     sp, x0, #0x2b0
ffff8000100108b0:       d538d080        mrs     x0, tpidr_el1
ffff8000100108b4:       8b2063ff        add     sp, sp, x0
ffff8000100108b8:       d53bd040        mrs     x0, tpidr_el0
ffff8000100108bc:       cb2063e0        sub     x0, sp, x0
ffff8000100108c0:       f274cc1f        tst     x0, #0xfffffffffffff000
ffff8000100108c4:       540038a1        b.ne    ffff800010010fd8 <__bad_stack>  // b.any
ffff8000100108c8:       cb2063ff        sub     sp, sp, x0
ffff8000100108cc:       d53bd060        mrs     x0, tpidrro_el0
ffff8000100108d0:       140003df        b       ffff80001001184c <el1_irq_invalid>
ffff8000100108d4:       d503201f        nop
ffff8000100108d8:       d503201f        nop
ffff8000100108dc:       d503201f        nop
ffff8000100108e0:       d503201f        nop
```

System IRQ handling Process

el1_irq()->irq_handler()->**handle_arch_irq()**

NOTE: handle_arch_irq is Top level irq for an ARCH.

For ARM64, usually set by system irqchip driver. e.g. gic

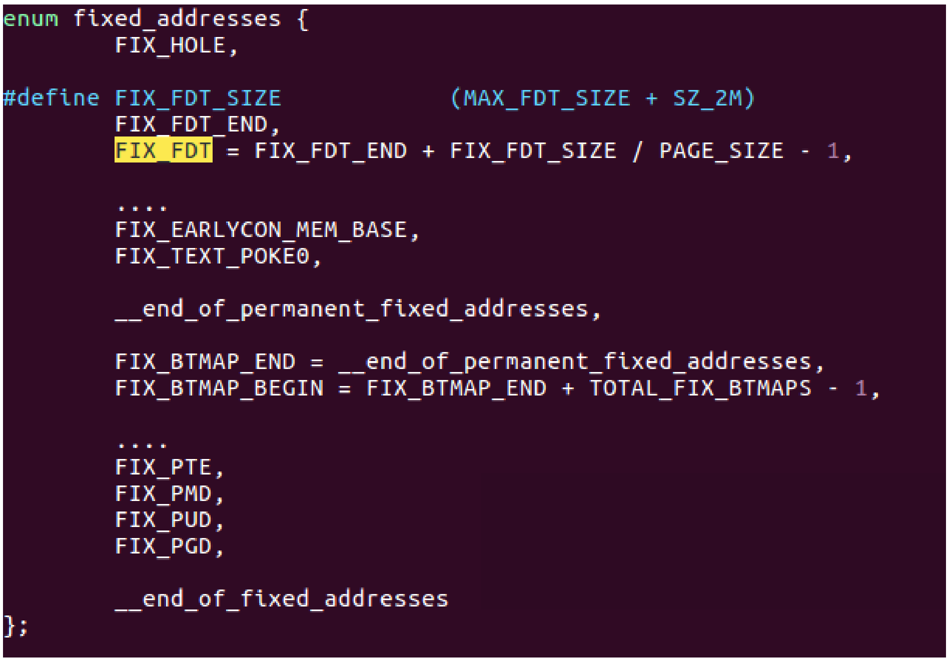
```
int __init set_handle_irq(void (*handle_irq)(struct pt_regs *))
{
        if (handle_arch_irq)
                return -EBUSY;

        handle_arch_irq = handle_irq;
        return 0;
}
```

drivers/irqchip/irq-gic-v3.c

gic_init_bases() -> set_handle_irq(gic_handle_irq);

gic_handle_irq() -> **handle_domain_irq(irq_domain, hwirq, regs)** -> irq handler or **route to the next irq_domain handler**

4. FDT Mapping



```
enum fixed_addresses {
        FIX_HOLE,

#define FIX_FDT_SIZE            (MAX_FDT_SIZE + SZ_2M)
        FIX_FDT_END,
        FIX_FDT = FIX_FDT_END + FIX_FDT_SIZE / PAGE_SIZE - 1,

        ....
        FIX_EARLYCON_MEM_BASE,
        FIX_TEXT_POKE0,

        __end_of_permanent_fixed_addresses,

        FIX_BTMAP_END = __end_of_permanent_fixed_addresses,
        FIX_BTMAP_BEGIN = FIX_BTMAP_END + TOTAL_FIX_BTMAPS - 1,

        ....
        FIX_PTE,
        FIX_PMD,
        FIX_PUD,
        FIX_PGD,

        __end_of_fixed_addresses
};
```

- Page table: init_pg_dir
- VA: FIX_FDT
- PA: dt_phys passed by bootloader
- NOTE: can’t exceed 2M
- #define FIXADDR_TOP (PCI_IO_START - SZ_2M)
- #define __fix_to_virt(x) (FIXADDR_TOP - ((x) << PAGE_SHIFT))
- dt_virt_base = __fix_to_virt(FIX_FDT);

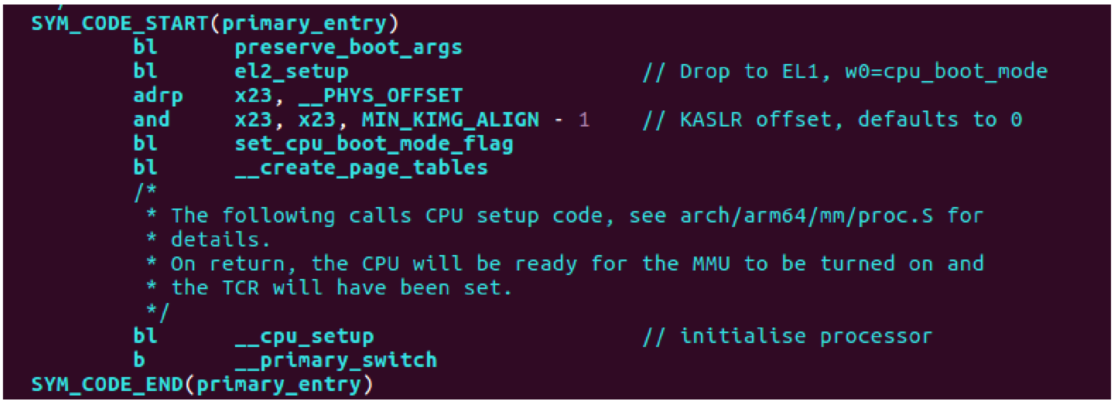
5. ARM64 Kernel booting process

5.1 Prior to start_kernel

First instruction in kernel

b primary_entry // branch to kernel start, magic

Major work prior to start kernel



```
SYM_CODE_START(primary_entry)
        bl      preserve_boot_args
        bl      el2_setup                       // Drop to EL1, w0=cpu_boot_mode
        adrp    x23, __PHYS_OFFSET
        and     x23, x23, MIN_KIMG_ALIGN - 1    // KASLR offset, defaults to 0
        bl      set_cpu_boot_mode_flag
        bl      __create_page_tables
        /*
         * The following calls CPU setup code, see arch/arm64/mm/proc.S for
         * details.
         * On return, the CPU will be ready for the MMU to be turned on and
         * the TCR will have been set.
         */
        bl      __cpu_setup                     // initialise processor
        b       __primary_switch
SYM_CODE_END(primary_entry)
```

5.1.1__create_page_tables

(1)Identity mapping for MMU enablement code

Page table: idmap_pg_dir (3 pages pre-allocated in vmlinux.lds.S)

VA: Runtime __pa of section ".idmap.text"

PA: Runtime __pa of section ".idmap.text“

Note: section ".idmap.text" includes MMU on code

(2) Kernel Image Mapping

Page table: init_pg_dir

VA: KIMAGE_VADDR / Compile time __va(text)

PA: Runtime __pa(_text) in DRAM

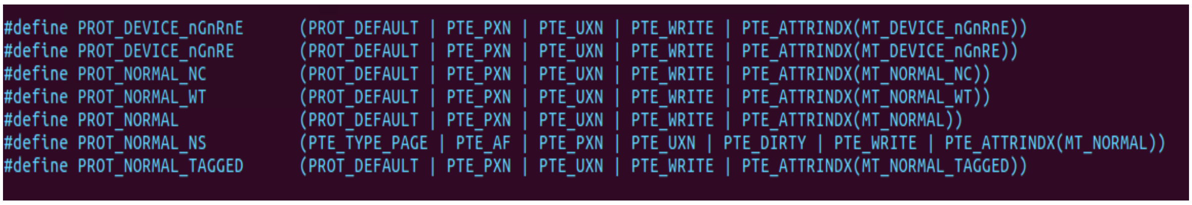
NOTE: ‘text’ section is in vmalloc address range

5.1.2 __cpu_setup

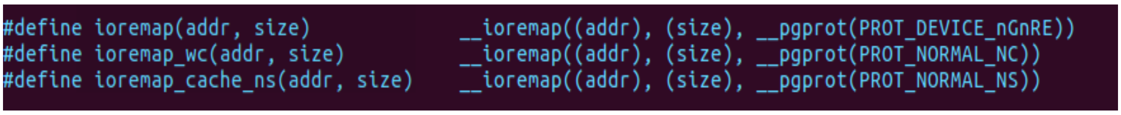
(1) Invalidate local TLB

(2) Disable PMU/AMU access from EL0

(3) Memory region attributes



```
#define PROT_DEVICE_nGnRnE      (PROT_DEFAULT | PTE_PXN | PTE_UXN | PTE_WRITE | PTE_ATTRINDX(MT_DEVICE_nGnRnE))
#define PROT_DEVICE_nGnRE       (PROT_DEFAULT | PTE_PXN | PTE_UXN | PTE_WRITE | PTE_ATTRINDX(MT_DEVICE_nGnRE))
#define PROT_NORMAL_NC          (PROT_DEFAULT | PTE_PXN | PTE_UXN | PTE_WRITE | PTE_ATTRINDX(MT_NORMAL_NC))
#define PROT_NORMAL_WT          (PROT_DEFAULT | PTE_PXN | PTE_UXN | PTE_WRITE | PTE_ATTRINDX(MT_NORMAL_WT))
#define PROT_NORMAL             (PROT_DEFAULT | PTE_PXN | PTE_UXN | PTE_WRITE | PTE_ATTRINDX(MT_NORMAL))
#define PROT_NORMAL_NS          (PTE_TYPE_PAGE | PTE_AF | PTE_PXN | PTE_UXN | PTE_DIRTY | PTE_WRITE | PTE_ATTRINDX(MT_NORMAL))
#define PROT_NORMAL_TAGGED      (PROT_DEFAULT | PTE_PXN | PTE_UXN | PTE_WRITE | PTE_ATTRINDX(MT_NORMAL_TAGGED))
```



```
#define ioremap(addr, size)             __ioremap((addr), (size), __pgprot(PROT_DEVICE_nGnRE))
#define ioremap_wc(addr, size)          __ioremap((addr), (size), __pgprot(PROT_NORMAL_NC))
#define ioremap_cache_ns(addr, size)    __ioremap((addr), (size), __pgprot(PROT_NORMAL_NS))
```

（4）**48-bit address range and 4K page table setting**

5.1.3 __primary_switch

（1）**__enable_mmu**

TTBR0: idmap_pg_dir

TTBR1: init_pg_dir

(2) Kernel image address randomization setting (KASLR)

(3) Setup kernel stack, thread_info/init_task

(4) Load VBAR_EL1 with virtual vector table address

(5) Calculate kimage_voffset for supporing __pa(x)/__pa_symbol(x)

(6) Clear BSS

(7) Create FDT mapping (*see next slides*)

(8) Call into start_kernel()

5.2 Start_kernel

(1) Architecture Setup (e.g. setup_arch())

(2) Memory Subsystem init

Memory zones

Memory buddy system

(3) Schedule init

(4) IRQ init

of_irq_init(__irqchip_of_table)

Driver: IRQCHIP_DECLARE(gic_v3, "arm,gic-v3", gic_of_init);

(5) Timer init

Clocks/clocksource/clockevent/cyclecounter register.

of_clk_init(NULL);

Driver: CLK_OF_DECLARE(imx7ulp_clk_scg1, "fsl,imx7ulp-scg1", imx7ulp_clk_scg1_init);

TIMER_OF_DECLARE(armv8_arch_timer, "arm,armv8-timer", arch_timer_of_init);

(6) Console init

(7) Other core functions init

E.g. cgroup_init() / kcsan_init()

(8) Reset Init

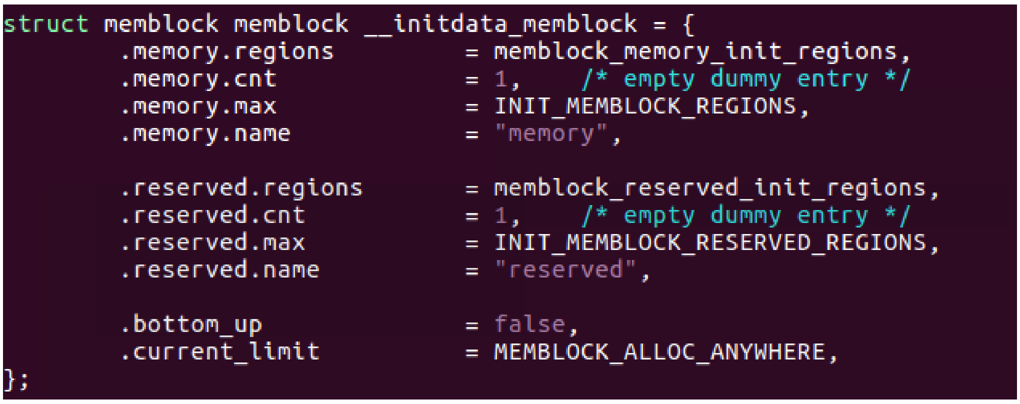
5.2.1 Start_kernel -> setup_arch

- **early_ioremap_init for early users of early_ioremap(paddr, size)**
- **setup_machine_fdt**
- **parse_early_param**
  - **early_param("mem", early_mem);**
  - **early_param("earlycon", param_setup_earlycon);**
  - **early_param("debug", debug_kernel);**
- **cpu_uninstall_idmap**
- **arm64_memblock_init**
  - **Reserve memory used by kernel image**
  - **Reserve memory specified in DT and specifical initialization if any**

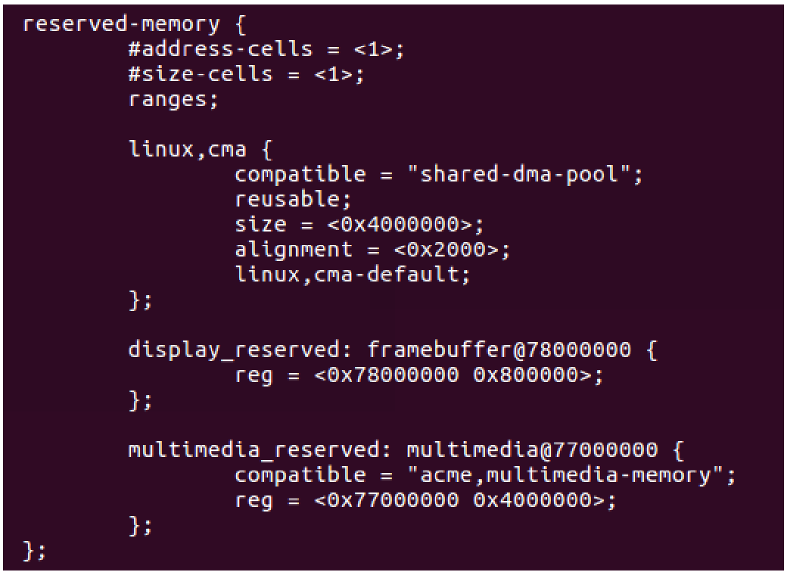
e.g. RESERVEDMEM_OF_DECLARE(cma, "shared-dma-pool", rmem_cma_setup);

- **unflatten_device_tree**

paging_init and bootmem_init



```
struct memblock memblock __initdata_memblock = {
        .memory.regions         = memblock_memory_init_regions,
        .memory.cnt             = 1,    /* empty dummy entry */
        .memory.max             = INIT_MEMBLOCK_REGIONS,
        .memory.name            = "memory",

        .reserved.regions       = memblock_reserved_init_regions,
        .reserved.cnt           = 1,    /* empty dummy entry */
        .reserved.max           = INIT_MEMBLOCK_RESERVED_REGIONS,
        .reserved.name          = "reserved",

        .bottom_up              = false,
        .current_limit          = MEMBLOCK_ALLOC_ANYWHERE,
};
```



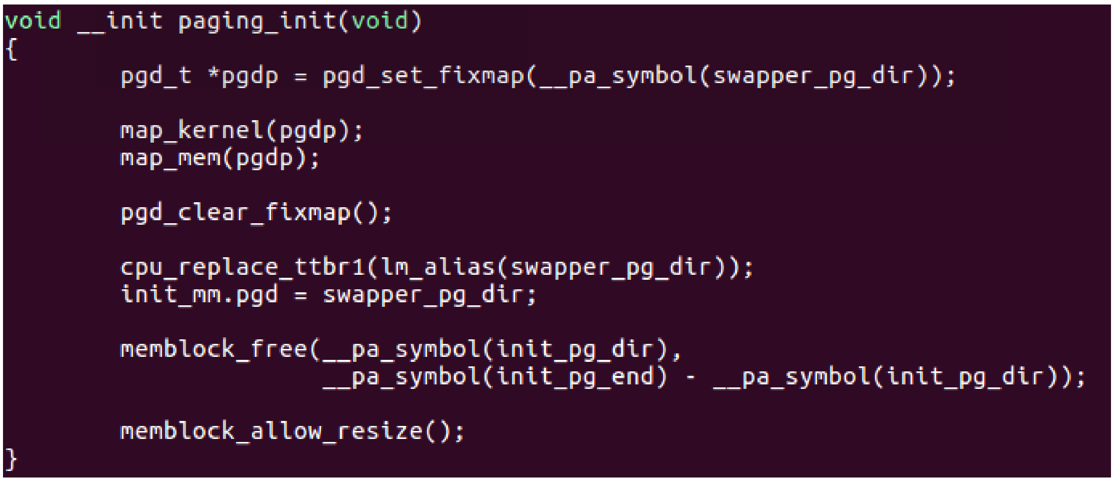
```
reserved-memory {
        #address-cells = <1>;
        #size-cells = <1>;
        ranges;

        linux,cma {
                compatible = "shared-dma-pool";
                reusable;
                size = <0x4000000>;
                alignment = <0x2000>;
                linux,cma-default;
        };

        display_reserved: framebuffer@78000000 {
                reg = <0x78000000 0x800000>;
        };

        multimedia_reserved: multimedia@77000000 {
                compatible = "acme,multimedia-memory";
                reg = <0x77000000 0x4000000>;
        };
};
```

5.2.1.1 Start_kernel -> setup_arch -> setup_machine_fdt

- **Parse ‘bootargs’ from DT ‘chosen’ node**
- **Parse Physical Memory base and size, added into memblock subsystem**
- **Parse Machine model**

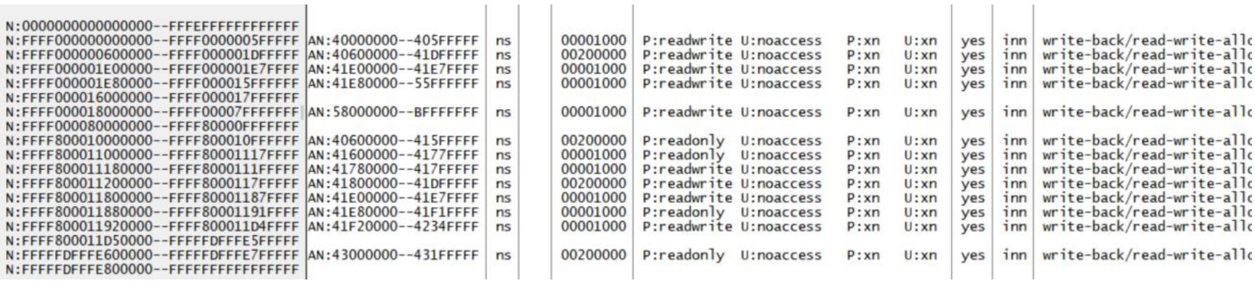
5.2.1.2 Start_kernel -> setup_arch -> paging_init / bootmem_init

- **Remap kernel sections _text, _rodata, _data and etc with different permissions**
- **Linear mapping for available physical memory blocks**
  #define __phys_to_virt(x)   ((unsigned long)((x) - PHYS_OFFSET) | PAGE_OFFSET)
- **Switch to swapper_pg_dir**
- **Build structure pages / vmemmap**
  - **Sparse_init**
- **Build memory zones**
  - **Usually only one DMA zone for ARM64**



```
void __init paging_init(void)
{
        pgd_t *pgdp = pgd_set_fixmap(__pa_symbol(swapper_pg_dir));

        map_kernel(pgdp);
        map_mem(pgdp);

        pgd_clear_fixmap();

        cpu_replace_ttbr1(lm_alias(swapper_pg_dir));
        init_mm.pgd = swapper_pg_dir;

        memblock_free(__pa_symbol(init_pg_dir),
                      __pa_symbol(init_pg_end) - __pa_symbol(init_pg_dir));

        memblock_allow_resize();
}
```

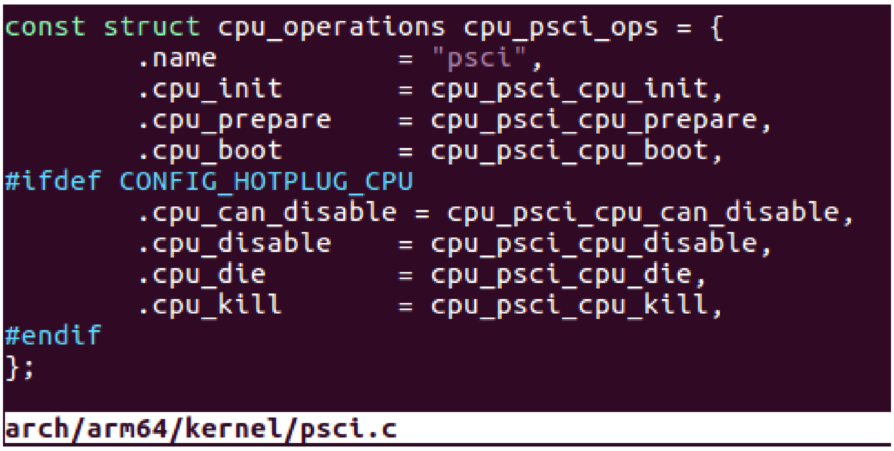
Page table dump after paging_init:



```
N:0000000000000000--FFFEFFFFFFFFFFFF
N:FFFF000000000000--FFFF0000005FFFFF AN:40000000--405FFFFF  ns   00001000 P:readwrite U:noaccess  P:xn  U:xn  yes inn write-back/read-write-all
N:FFFF000000600000--FFFF000001DFFFFF AN:40600000--41DFFFFF  ns   00200000 P:readwrite U:noaccess  P:xn  U:xn  yes inn write-back/read-write-all
N:FFFF000001E00000--FFFF000001E7FFFF AN:41E00000--41E7FFFF  ns   00001000 P:readwrite U:noaccess  P:xn  U:xn  yes inn write-back/read-write-all
N:FFFF000001E80000--FFFF000015FFFFFF AN:41E80000--55FFFFFF  ns   00001000 P:readwrite U:noaccess  P:xn  U:xn  yes inn write-back/read-write-all
N:FFFF000016000000--FFFF000017FFFFFF
N:FFFF000018000000--FFFF00007FFFFFFF AN:58000000--BFFFFFFF  ns   00001000 P:readwrite U:noaccess  P:xn  U:xn  yes inn write-back/read-write-all
N:FFFF000080000000--FFFF80000FFFFFFF
N:FFFF800010000000--FFFF800010FFFFFF AN:40600000--415FFFFF  ns   00200000 P:readonly  U:noaccess  P:xn  U:xn  yes inn write-back/read-write-all
N:FFFF800011000000--FFFF80001117FFFF AN:41600000--4177FFFF  ns   00001000 P:readonly  U:noaccess  P:xn  U:xn  yes inn write-back/read-write-all
N:FFFF800011180000--FFFF8000111FFFFF AN:41780000--417FFFFF  ns   00001000 P:readwrite U:noaccess  P:xn  U:xn  yes inn write-back/read-write-all
N:FFFF800011200000--FFFF8000117FFFFF AN:41800000--41DFFFFF  ns   00200000 P:readwrite U:noaccess  P:xn  U:xn  yes inn write-back/read-write-all
N:FFFF800011800000--FFFF80001187FFFF AN:41E00000--41E7FFFF  ns   00001000 P:readwrite U:noaccess  P:xn  U:xn  yes inn write-back/read-write-all
N:FFFF800011880000--FFFF80001191FFFF AN:41E80000--41F1FFFF  ns   00001000 P:readonly  U:noaccess  P:xn  U:xn  yes inn write-back/read-write-all
N:FFFF800011920000--FFFF800011D4FFFF AN:41F20000--4234FFFF  ns   00001000 P:readwrite U:noaccess  P:xn  U:xn  yes inn write-back/read-write-all
N:FFFF800011D50000--FFFFFDFFFE5FFFFF
N:FFFFFDFFFE600000--FFFFFDFFFE7FFFFF AN:43000000--431FFFFF  ns   00200000 P:readonly  U:noaccess  P:xn  U:xn  yes inn write-back/read-write-all
N:FFFFFDFFFE800000--FFFFFFFFFFFFFFFF
```

5.2.1.3 Start_kernel -> setup_arch -> psci_init

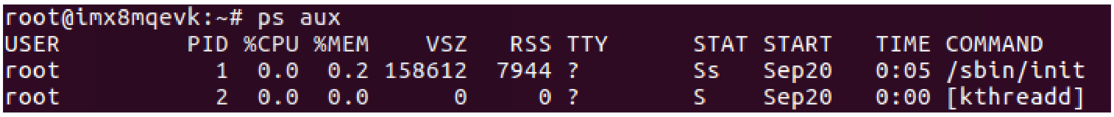
Firmware interface implementing CPU power related operations specified by ARM PSCI spec
Including CPU_ON/OFF/SUSPNED/MIGRATION and etc.



```
const struct cpu_operations cpu_psci_ops = {
        .name           = "psci",
        .cpu_init       = cpu_psci_cpu_init,
        .cpu_prepare    = cpu_psci_cpu_prepare,
        .cpu_boot       = cpu_psci_cpu_boot,
#ifdef CONFIG_HOTPLUG_CPU
        .cpu_can_disable = cpu_psci_cpu_can_disable,
        .cpu_disable    = cpu_psci_cpu_disable,
        .cpu_die        = cpu_psci_cpu_die,
        .cpu_kill       = cpu_psci_cpu_kill,
#endif
};

arch/arm64/kernel/psci.c
```

5.2.2 Start_kernel -> Rest_init

- Populate the first three kernel threads
  - PID 0 -> Idle thread per CPU
    - cpu_startup_entry -> do_idle -> cpuidle_idle_call -> cpuidle_enter ->
    - cpuidle_enter_state (selected by governor) -> psci suspend state (cpuidle-psci.c) -> TF-A
  - PID 1 -> Init thread
    - Keep running kernel_init() for the rest kernel initialization work
  - PID 2 -> kthreadd
    - Used for kthread_run(hwrng_fillfn, NULL, "hwrng");
  - Userspace dump



```
root@imx8mqevk:~# ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.2 158612  7944 ?        Ss   Sep20   0:05 /sbin/init
root         2  0.0  0.0      0     0 ?        S    Sep20   0:00 [kthreadd]
```

5.2.2.1 Start_kernel -> Rest_init -> kernel_init

- SMP init
  - Bring up non-boot CPUs

    bringup_nonboot_cpus -> cpuhp_up_callbacks -> boot_secondary -> cpu_psci_cpu_boot -> TF-A
  - Difference from Booting CPU

    Mainly are same except no __create_page_tables

- do_initcalls
  - Usually used for device and driver register

    e.g. module_init()
  - Call in the order below:

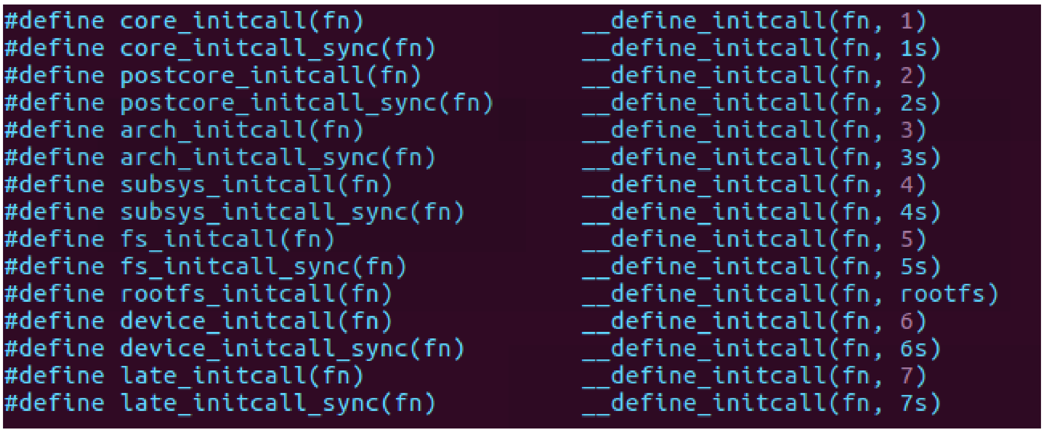
    __initcall0_start,

    …

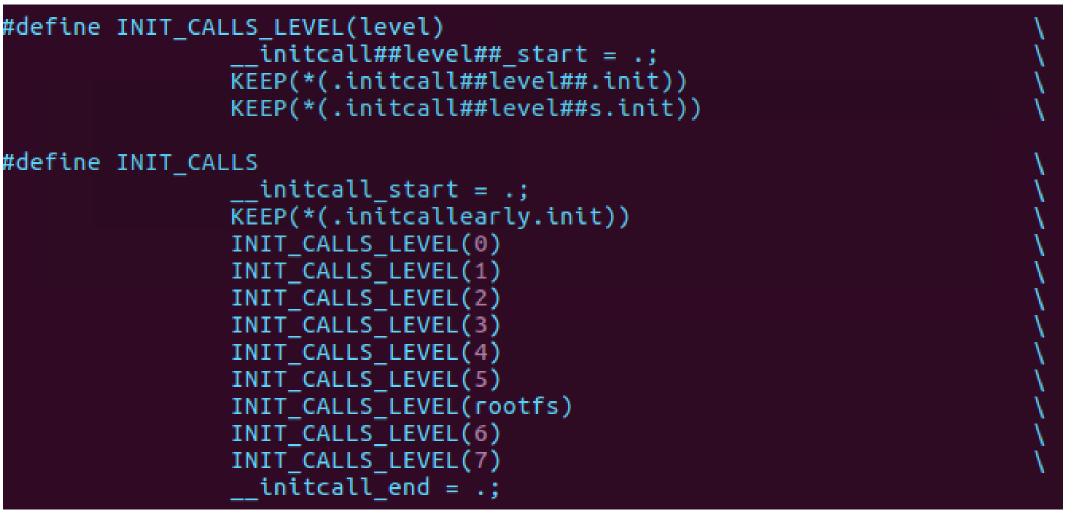
    __initcall7_start,
  - Become invalid when build as module

    All default to module_init()
  - Platform devices populated in arch level

    e.g.arch_initcall_sync(of_platform_default_populate_init);



```
#define core_initcall(fn)               __define_initcall(fn, 1)
#define core_initcall_sync(fn)          __define_initcall(fn, 1s)
#define postcore_initcall(fn)           __define_initcall(fn, 2)
#define postcore_initcall_sync(fn)      __define_initcall(fn, 2s)
#define arch_initcall(fn)               __define_initcall(fn, 3)
#define arch_initcall_sync(fn)          __define_initcall(fn, 3s)
#define subsys_initcall(fn)             __define_initcall(fn, 4)
#define subsys_initcall_sync(fn)        __define_initcall(fn, 4s)
#define fs_initcall(fn)                 __define_initcall(fn, 5)
#define fs_initcall_sync(fn)            __define_initcall(fn, 5s)
#define rootfs_initcall(fn)             __define_initcall(fn, rootfs)
#define device_initcall(fn)             __define_initcall(fn, 6)
#define device_initcall_sync(fn)        __define_initcall(fn, 6s)
#define late_initcall(fn)               __define_initcall(fn, 7)
#define late_initcall_sync(fn)          __define_initcall(fn, 7s)
```



```
#define INIT_CALLS_LEVEL(level)                                         \
                __initcall##level##_start = .;                          \
                KEEP(*(.initcall##level##.init))                        \
                KEEP(*(.initcall##level##s.init))                       \

#define INIT_CALLS                                                      \
                __initcall_start = .;                                   \
                KEEP(*(.initcallearly.init))                            \
                INIT_CALLS_LEVEL(0)                                     \
                INIT_CALLS_LEVEL(1)                                     \
                INIT_CALLS_LEVEL(2)                                     \
                INIT_CALLS_LEVEL(3)                                     \
                INIT_CALLS_LEVEL(4)                                     \
                INIT_CALLS_LEVEL(5)                                     \
                INIT_CALLS_LEVEL(rootfs)                                \
                INIT_CALLS_LEVEL(6)                                     \
                INIT_CALLS_LEVEL(7)                                     \
                __initcall_end = .;
```

- Filp_open(/dev/console)
- Mount rootfs in prepare_namespace
- Free init memory between __init_begin and __init_end
- Run the first userpace application in the order

  execute_command. E.g. init=/bin/sh

  CONFIG_DEFAULT_INIT

  /sbin/init, /etc/init, /bin/init, /bin/sh