Linux Riscv Documentation

The kernel development community

Jan 15, 2023

CONTENTS

CHAPTER

ONE

BOOT IMAGE HEADER IN RISC-V LINUX

Author Atish Patra <atish.patra@wdc.com>

Date 20 May 2019

This document only describes the boot image header details for RISC-V Linux.

TODO: Write a complete booting guide.

The following 64-byte header is present in decompressed Linux kernel image:

```
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 */
u32 version;              /* Version of this header */
u32 res1 = 0;             /* Reserved */
u64 res2 = 0;             /* Reserved */
u64 magic = 0x5643534952; /* Magic number, little endian, "RISCV" */
u32 magic2 = 0x05435352;  /* Magic number 2, little endian, "RSC\x05" */
u32 res3;                 /* Reserved for PE COFF offset */
```

This header format is compliant with PE/COFF header and largely inspired from ARM64 header. Thus, both ARM64 & RISC-V header can be combined into one common header in future.

1.1 Notes

- This header can also be reused to support EFI stub for RISC-V in future. EFI specification needs PE/COFF image header in the beginning of the kernel image in order to load it as an EFI application. In order to support EFI stub, code0 should be replaced with “MZ” magic string and res3(at offset 0x3c) should point to the rest of the PE/COFF header.
- version field indicate header version number

Bits 0:15	Minor version
Bits 16:31	Major version

  This preserves compatibility across newer and older version of the header. The current version is defined as 0.2.

- The “magic” field is deprecated as of version 0.2. In a future release, it may be removed. This originally should have matched up with the ARM64 header “magic” field, but unfortunately does not. The “magic2” field replaces it, matching up with the ARM64 header.
- In current header, the flags field has only one field.

Bit 0	Kernel endianness. 1 if BE, 0 if LE.

- Image size is mandatory for boot loader to load kernel image. Booting will fail otherwise.

CHAPTER

TWO

VIRTUAL MEMORY LAYOUT ON RISC-V LINUX

Author Alexandre Ghiti <alex@ghiti.fr>

Date 12 February 2021

This document describes the virtual memory layout used by the RISC-V Linux Kernel.

2.1 RISC-V Linux Kernel 32bit

2.1.1 RISC-V Linux Kernel SV32

TODO

2.2 RISC-V Linux Kernel 64bit

The RISC-V privileged architecture document states that the 64bit addresses "must have bits 63–48 all equal to bit 47, or else a page-fault exception will occur.": that splits the virtual address space into 2 halves separated by a very big hole, the lower half is where the userspace resides, the upper half is where the RISC-V Linux Kernel resides.

2.2.1 RISC-V Linux Kernel SV39

```
========================================================================================
    Start addr    |   Offset   |     End addr     |  Size   | VM area␣
↪description
========================================================================================
                  |            |                  |         |
 0000000000000000 |    0       | 0000003fffffffff |  256 GB | user-space␣
↪virtual memory, different per mm
__________________|____________|__________________|_________|___________________
↪_____________________________________
                  |            |                  |         |
 0000004000000000 | +256    GB | ffffffbfffffffff | ~16M TB | ... huge, almost␣
↪64 bits wide hole of non-canonical
                  |            |                  |         |     virtual␣
↪memory addresses up to the -256 GB
                  |            |                  |         |     starting␣
↪offset of kernel mappings.
```

```
____________________|____________|__________________|_________|____________________
↪_____________________________________________
                                                                |
                                                                | Kernel-space␣
↪virtual memory, shared between all processes:
________________________________________________________________|____________________
↪_____________________________________________
                    |            |                  |         |
 ffffffc6fee00000 | -228    GB | ffffffc6feffffff |    2 MB | fixmap
 ffffffc6ff000000 | -228    GB | ffffffc6ffffffff |   16 MB | PCI io
 ffffffc700000000 | -228    GB | ffffffc7ffffffff |    4 GB | vmemmap
 ffffffc800000000 | -224    GB | ffffffd7ffffffff |   64 GB | vmalloc/ioremap␣
↪space
 ffffffd800000000 | -160    GB | fffffff6ffffffff |  124 GB | direct mapping␣
↪of all physical memory
 fffffff700000000 |  -36    GB | fffffffeffffffff |   32 GB | kasan
____________________|____________|__________________|_________|____________________
↪_____________________________________________
                                                                |
                                                                |
________________________________________________________________|____________________
↪_____________________________________________
                    |            |                  |         |
 ffffffff00000000 |   -4    GB | ffffffff7fffffff |    2 GB | modules, BPF
 ffffffff80000000 |   -2    GB | ffffffffffffffff |    2 GB | kernel
____________________|____________|__________________|_________|____________________
↪_____________________________________________
```

2.2.2 RISC-V Linux Kernel SV48

```
================================================================================================
    Start addr    |   Offset   |     End addr     |  Size   | VM area␣
↪description
================================================================================================
                  |            |                  |         |
  0000000000000000 |    0       | 00007fffffffffff |  128 TB | user-space␣
↪virtual memory, different per mm
 _________________|____________|__________________|_________|____________________
↪_____________________________________________
                  |            |                  |         |
  0000800000000000 | +128    TB | ffff7fffffffffff | ~16M TB | ... huge,␣
↪almost 64 bits wide hole of non-canonical
                  |            |                  |         | virtual memory␣
↪addresses up to the -128 TB
                  |            |                  |         | starting offset␣
↪of kernel mappings.
 _________________|____________|__________________|_________|____________________
↪_____________________________________________
                                                             |
```

```
                                                               | Kernel-space␣
↪virtual memory, shared between all processes:
_____________________________________________________________|___________________
↪_____________________________________________
                    |             |                  |          |
 ffff8d7ffee00000 |  -114.5 TB | ffff8d7ffeffffff |    2 MB | fixmap
 ffff8d7fff000000 |  -114.5 TB | ffff8d7fffffffff |   16 MB | PCI io
 ffff8d8000000000 |  -114.5 TB | ffff8f7fffffffff |    2 TB | vmemmap
 ffff8f8000000000 |  -112.5 TB | ffffaf7fffffffff |   32 TB | vmalloc/ioremap␣
↪space
 ffffaf8000000000 |  -80.5  TB | ffffef7fffffffff |   64 TB | direct mapping␣
↪of all physical memory
 ffffef8000000000 |  -16.5  TB | fffffffeffffffff | 16.5 TB | kasan
_____________________|_____________|__________________|_________|___________________
↪_____________________________________________
                                                               |
                                                               | Identical␣
↪layout to the 39-bit one from here on:
_____________________________________________________________|___________________
↪_____________________________________________
                    |             |                  |          |
 ffffffff00000000 |    -4    GB | ffffffff7fffffff |    2 GB | modules, BPF
 ffffffff80000000 |    -2    GB | ffffffffffffffff |    2 GB | kernel
_____________________|_____________|__________________|_________|___________________
↪_____________________________________________
```

CHAPTER

THREE

ARCH/RISCV MAINTENANCE GUIDELINES FOR DEVELOPERS

3.1 Overview

The RISC-V instruction set architecture is developed in the open: in-progress drafts are available for all to review and to experiment with implementations. New module or extension drafts can change during the development process - sometimes in ways that are incompatible with previous drafts. This flexibility can present a challenge for RISC-V Linux maintenance. Linux maintainers disapprove of churn, and the Linux development process prefers well-reviewed and tested code over experimental code. We wish to extend these same principles to the RISC-V-related code that will be accepted for inclusion in the kernel.

3.2 Submit Checklist Addendum

We'll only accept patches for new modules or extensions if the specifications for those modules or extensions are listed as being "Frozen" or "Ratified" by the RISC-V Foundation. (Developers may, of course, maintain their own Linux kernel trees that contain code for any draft extensions that they wish.)

Additionally, the RISC-V specification allows implementors to create their own custom extensions. These custom extensions aren't required to go through any review or ratification process by the RISC-V Foundation. To avoid the maintenance complexity and potential performance impact of adding kernel code for implementor-specific RISC-V extensions, we'll only to accept patches for extensions that have been officially frozen or ratified by the RISC-V Foundation. (Implementors, may, of course, maintain their own Linux kernel trees containing code for any custom extensions that they wish.)

CHAPTER

FOUR

FEATURE STATUS ON RISCV ARCHITECTURE

Subsystem	Feature	Kconfig	Status	Desc
core	cBPF-JIT	HAVE_CBPF_JIT	TODO	arch
core	eBPF-JIT	HAVE_EBPF_JIT	ok	arch
core	generic-idle-thread	GENERIC_SMP_IDLE_THREAD	ok	arch
core	jump-labels	HAVE_ARCH_JUMP_LABEL	ok	arch
core	thread-info-in-task	THREAD_INFO_IN_TASK	ok	arch
core	tracehook	HAVE_ARCH_TRACEHOOK	ok	arch
debug	debug-vm-pgtable	ARCH_HAS_DEBUG_VM_PGTABLE	ok	arch
debug	gcov-profile-all	ARCH_HAS_GCOV_PROFILE_ALL	ok	arch
debug	KASAN	HAVE_ARCH_KASAN	ok	arch
debug	kcov	ARCH_HAS_KCOV	ok	arch
debug	kgdb	HAVE_ARCH_KGDB	ok	arch
debug	kmemleak	HAVE_DEBUG_KMEMLEAK	ok	arch
debug	kprobes	HAVE_KPROBES	ok	arch
debug	kprobes-on-ftrace	HAVE_KPROBES_ON_FTRACE	ok	arch
debug	kretprobes	HAVE_KRETPROBES	ok	arch
debug	optprobes	HAVE_OPTPROBES	TODO	arch
debug	stackprotector	HAVE_STACKPROTECTOR	ok	arch
debug	uprobes	ARCH_SUPPORTS_UPROBES	ok	arch
debug	user-ret-profiler	HAVE_USER_RETURN_NOTIFIER	TODO	arch
io	dma-contiguous	HAVE_DMA_CONTIGUOUS	ok	arch
locking	cmpxchg-local	HAVE_CMPXCHG_LOCAL	TODO	arch
locking	lockdep	LOCKDEP_SUPPORT	ok	arch
locking	queued-rwlocks	ARCH_USE_QUEUED_RWLOCKS	ok	arch
locking	queued-spinlocks	ARCH_USE_QUEUED_SPINLOCKS	TODO	arch
perf	kprobes-event	HAVE_REGS_AND_STACK_ACCESS_API	ok	arch
perf	perf-regs	HAVE_PERF_REGS	ok	arch
perf	perf-stackdump	HAVE_PERF_USER_STACK_DUMP	ok	arch
sched	membarrier-sync-core	ARCH_HAS_MEMBARRIER_SYNC_CORE	TODO	arch
sched	numa-balancing	ARCH_SUPPORTS_NUMA_BALANCING	ok	arch
seccomp	seccomp-filter	HAVE_ARCH_SECCOMP_FILTER	ok	arch
time	arch-tick-broadcast	ARCH_HAS_TICK_BROADCAST	ok	arch
time	clockevents	!LEGACY_TIMER_TICK	ok	arch
time	context-tracking	HAVE_CONTEXT_TRACKING	ok	arch
time	irq-time-acct	HAVE_IRQ_TIME_ACCOUNTING	ok	arch
time	virt-cpuacct	HAVE_VIRT_CPU_ACCOUNTING	TODO	arch

Table 1 – continued from p

Subsystem	Feature	Kconfig	Status	Desc
vm	batch-unmap-tlb-flush	ARCH_WANT_BATCHED_UNMAP_TLB_FLUSH	TODO	arch
vm	ELF-ASLR	ARCH_HAS_ELF_RANDOMIZE	ok	arch
vm	huge-vmap	HAVE_ARCH_HUGE_VMAP	TODO	arch
vm	ioremap_prot	HAVE_IOREMAP_PROT	TODO	arch
vm	PG_uncached	ARCH_USES_PG_UNCACHED	TODO	arch
vm	pte_special	ARCH_HAS_PTE_SPECIAL	ok	arch
vm	THP	HAVE_ARCH_TRANSPARENT_HUGEPAGE	ok	arch