Linux Riscv Documentation

The kernel development community

Jun 10, 2024

CONTENTS

CHAPTER

ONE

ACPI ON RISC-V

The ISA string parsing rules for ACPI are defined by Version ASCIIDOC Conversion, 12/2022 of the RISC-V specifications, as defined by tag “riscv-isa-release-1239329-2023-05-23” (commit 1239329)

CHAPTER

TWO

RISC-V KERNEL BOOT REQUIREMENTS AND CONSTRAINTS

Author
Alexandre Ghiti <alexghiti@rivosinc.com>

Date
23 May 2023

This document describes what the RISC-V kernel expects from bootloaders and firmware, and also the constraints that any developer must have in mind when touching the early boot process. For the purposes of this document, the `early boot process` refers to any code that runs before the final virtual mapping is set up.

2.1 Pre-kernel Requirements and Constraints

The RISC-V kernel expects the following of bootloaders and platform firmware:

2.1.1 Register state

The RISC-V kernel expects:

- `$a0` to contain the hartid of the current core.
- `$a1` to contain the address of the devicetree in memory.

2.1.2 CSR state

The RISC-V kernel expects:

- `$satp = 0`: the MMU, if present, must be disabled.

2.1.3 Reserved memory for resident firmware

The RISC-V kernel must not map any resident memory, or memory protected with PMPs, in the direct mapping, so the firmware must correctly mark those regions as per the device-tree specification and/or the UEFI specification.

2.1.4 Kernel location

The RISC-V kernel expects to be placed at a PMD boundary (2MB aligned for rv64 and 4MB aligned for rv32). Note that the EFI stub will physically relocate the kernel if that's not the case.

2.1.5 Hardware description

The firmware can pass either a devicetree or ACPI tables to the RISC-V kernel.

The devicetree is either passed directly to the kernel from the previous stage using the `$a1` register, or when booting with UEFI, it can be passed using the EFI configuration table.

The ACPI tables are passed to the kernel using the EFI configuration table. In this case, a tiny devicetree is still created by the EFI stub. Please refer to "EFI stub and devicetree" section below for details about this devicetree.

2.1.6 Kernel entry

On SMP systems, there are 2 methods to enter the kernel:

- `RISCV_BOOT_SPINWAIT`: the firmware releases all harts in the kernel, one hart wins a lottery and executes the early boot code while the other harts are parked waiting for the initialization to finish. This method is mostly used to support older firmwares without SBI HSM extension and M-mode RISC-V kernel.
- `Ordered booting`: the firmware releases only one hart that will execute the initialization phase and then will start all other harts using the SBI HSM extension. The ordered booting method is the preferred booting method for booting the RISC-V kernel because it can support CPU hotplug and kexec.

2.1.7 UEFI

UEFI memory map

When booting with UEFI, the RISC-V kernel will use only the EFI memory map to populate the system memory.

The UEFI firmware must parse the subnodes of the `/reserved-memory` devicetree node and abide by the devicetree specification to convert the attributes of those subnodes (`no-map` and `reusable`) into their correct EFI equivalent (refer to section "3.5.4 /reserved-memory and UEFI" of the devicetree specification v0.4-rc1).

RISCV_EFI_BOOT_PROTOCOL

When booting with UEFI, the EFI stub requires the boot hartid in order to pass it to the RISC-V kernel in `$a1`. The EFI stub retrieves the boot hartid using one of the following methods:

- `RISCV_EFI_BOOT_PROTOCOL` (**preferred**).
- `boot-hartid` devicetree subnode (**deprecated**).

Any new firmware must implement `RISCV_EFI_BOOT_PROTOCOL` as the devicetree based approach is deprecated now.

2.2 Early Boot Requirements and Constraints

The RISC-V kernel's early boot process operates under the following constraints:

2.2.1 EFI stub and devicetree

When booting with UEFI, the devicetree is supplemented (or created) by the EFI stub with the same parameters as arm64 which are described at the paragraph "UEFI kernel support on ARM" in Documentation/arch/arm/uefi.rst.

2.2.2 Virtual mapping installation

The installation of the virtual mapping is done in 2 steps in the RISC-V kernel:

1. `setup_vm()` installs a temporary kernel mapping in `early_pg_dir` which allows discovery of the system memory. Only the kernel text/data are mapped at this point. When establishing this mapping, no allocation can be done (since the system memory is not known yet), so `early_pg_dir` page table is statically allocated (using only one table for each level).
2. `setup_vm_final()` creates the final kernel mapping in `swapper_pg_dir` and takes advantage of the discovered system memory to create the linear mapping. When establishing this mapping, the kernel can allocate memory but cannot access it directly (since the direct mapping is not present yet), so it uses temporary mappings in the fixmap region to be able to access the newly allocated page table levels.

For `virt_to_phys()` and `phys_to_virt()` to be able to correctly convert direct mapping addresses to physical addresses, they need to know the start of the DRAM. This happens after step 1, right before step 2 installs the direct mapping (see `setup_bootmem()` function in arch/riscv/mm/init.c). Any usage of those macros before the final virtual mapping is installed must be carefully examined.

2.2.3 Devicetree mapping via fixmap

As the `reserved_mem` array is initialized with virtual addresses established by `setup_vm()`, and used with the mapping established by `setup_vm_final()`, the RISC-V kernel uses the fixmap region to map the devicetree. This ensures that the devicetree remains accessible by both virtual mappings.

2.2.4 Pre-MMU execution

A few pieces of code need to run before even the first virtual mapping is established. These are the installation of the first virtual mapping itself, patching of early alternatives and the early parsing of the kernel command line. That code must be very carefully compiled as:

- `-fno-pie`: This is needed for relocatable kernels which use `-fPIE`, since otherwise, any access to a global symbol would go through the GOT which is only relocated virtually.
- `-mcmodel=medany`: Any access to a global symbol must be PC-relative to avoid any relocations to happen before the MMU is setup.
- *all* instrumentation must also be disabled (that includes KASAN, ftrace and others).

As using a symbol from a different compilation unit requires this unit to be compiled with those flags, we advise, as much as possible, not to use external symbols.

CHAPTER

THREE

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.

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.

3.1 Notes

- This header is also reused to support EFI stub for RISC-V. 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 is replaced with “MZ” magic string and res3(at offset 0x3c) points 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

FOUR

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.

4.1 RISC-V Linux Kernel 32bit

4.1.1 RISC-V Linux Kernel SV32

TODO

4.2 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.

4.2.1 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:
_____________________________________________________________|____________________
↪_____________________________________________
                  |            |                  |         |
 ffffffc6fea00000 | -228    GB | ffffffc6feffffff |    6 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
__________________|____________|__________________|_________|____________________
↪_____________________________________________
```

4.2.2 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:
________________________________________________________________|____________________
↪_____________________________________________
                    |            |                  |         |
 ffff8d7ffea00000 |  -114.5 TB | ffff8d7ffeffffff |    6 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
____________________|____________|__________________|_________|____________________
↪_____________________________________________
```

4.2.3 RISC-V Linux Kernel SV57

```
================================================================================================
    Start addr    |   Offset   |     End addr     |  Size   | VM area␣
↪description
================================================================================================
                    |            |                  |         |
 0000000000000000 |   0        | 00ffffffffffffff |   64 PB | user-space␣
↪virtual memory, different per mm
____________________|____________|__________________|_________|____________________
↪_____________________________________________
                    |            |                  |         |
 0100000000000000 | +64     PB | feffffffffffffff | ~16K PB | ... huge,␣
↪almost 64 bits wide hole of non-canonical
                    |            |                  |         | virtual memory␣
↪addresses up to the -64 PB
                    |            |                  |         | starting offset␣
↪of kernel mappings.
____________________|____________|__________________|_________|____________________
↪_____________________________________________
```

```
                                                                 |
                                                                 | Kernel-space␣
↪virtual memory, shared between all processes:
_________________________________________________________________|___________________
↪_____________________________________________
                    |             |                  |          |
  ff1bfffffea00000 | -57     PB | ff1bffffffeffffe |    6 MB | fixmap
  ff1bffffff000000 | -57     PB | ff1bffffffffffff |   16 MB | PCI io
  ff1c000000000000 | -57     PB | ff1fffffffffffff |    1 PB | vmemmap
  ff20000000000000 | -56     PB | ff5fffffffffffff |   16 PB | vmalloc/ioremap␣
↪space
  ff60000000000000 | -40     PB | ffdeffffffffffff |   32 PB | direct mapping␣
↪of all physical memory
  ffdf000000000000 |  -8     PB | fffffffeffffffff |    8 PB | 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
___________________|_____________|__________________|_________|___________________
↪_____________________________________________
```

4.2.4 Userspace VAs

To maintain compatibility with software that relies on the VA space with a maximum of 48 bits the kernel will, by default, return virtual addresses to userspace from a 48-bit range (sv48). This default behavior is achieved by passing 0 into the hint address parameter of mmap. On CPUs with an address space smaller than sv48, the CPU maximum supported address space will be the default.

Software can “opt-in” to receiving VAs from another VA space by providing a hint address to mmap. A hint address passed to mmap will cause the largest address space that fits entirely into the hint to be used, unless there is no space left in the address space. If there is no space available in the requested address space, an address in the next smallest available address space will be returned.

For example, in order to obtain 48-bit VA space, a hint address greater than `1 << 47` must be provided. Note that this is 47 due to sv48 userspace ending at `1 << 47` and the addresses beyond this are reserved for the kernel. Similarly, to obtain 57-bit VA space addresses, a hint address greater than or equal to `1 << 56` must be provided.

CHAPTER

FIVE

RISC-V HARDWARE PROBING INTERFACE

The RISC-V hardware probing interface is based around a single syscall, which is defined in <asm/hwprobe.h>:

```
struct riscv_hwprobe {
    __s64 key;
    __u64 value;
};

long sys_riscv_hwprobe(struct riscv_hwprobe *pairs, size_t pair_count,
                       size_t cpu_count, cpu_set_t *cpus,
                       unsigned int flags);
```

The arguments are split into three groups: an array of key-value pairs, a CPU set, and some flags. The key-value pairs are supplied with a count. Userspace must prepopulate the key field for each element, and the kernel will fill in the value if the key is recognized. If a key is unknown to the kernel, its key field will be cleared to -1, and its value set to 0. The CPU set is defined by CPU_SET(3). For value-like keys (eg. vendor/arch/impl), the returned value will be only be valid if all CPUs in the given set have the same value. Otherwise -1 will be returned. For boolean-like keys, the value returned will be a logical AND of the values for the specified CPUs. Usermode can supply NULL for cpus and 0 for cpu_count as a shortcut for all online CPUs. There are currently no flags, this value must be zero for future compatibility.

On success 0 is returned, on failure a negative error code is returned.

The following keys are defined:

- `RISCV_HWPROBE_KEY_MVENDORID`: Contains the value of `mvendorid`, as defined by the RISC-V privileged architecture specification.
- `RISCV_HWPROBE_KEY_MARCHID`: Contains the value of `marchid`, as defined by the RISC-V privileged architecture specification.
- `RISCV_HWPROBE_KEY_MIMPLID`: Contains the value of `mimplid`, as defined by the RISC-V privileged architecture specification.
- `RISCV_HWPROBE_KEY_BASE_BEHAVIOR`: A bitmask containing the base user-visible behavior that this kernel supports. The following base user ABIs are defined:
  - `RISCV_HWPROBE_BASE_BEHAVIOR_IMA`: Support for rv32ima or rv64ima, as defined by version 2.2 of the user ISA and version 1.10 of the privileged ISA, with the following known exceptions (more exceptions may be added, but only if it can be demonstrated that the user ABI is not broken):

    * The `fence.i` instruction cannot be directly executed by userspace programs (it may still be executed in userspace via a kernel-controlled mechanism such as the vDSO).

- `RISCV_HWPROBE_KEY_IMA_EXT_0`: A bitmask containing the extensions that are compatible with the `RISCV_HWPROBE_BASE_BEHAVIOR_IMA`: base system behavior.
  - `RISCV_HWPROBE_IMA_FD`: The F and D extensions are supported, as defined by commit cd20cee ("FMIN/FMAX now implement minimumNumber/maximumNumber, not minNum/maxNum") of the RISC-V ISA manual.
  - `RISCV_HWPROBE_IMA_C`: The C extension is supported, as defined by version 2.2 of the RISC-V ISA manual.
  - `RISCV_HWPROBE_IMA_V`: The V extension is supported, as defined by version 1.0 of the RISC-V Vector extension manual.
  - **`RISCV_HWPROBE_EXT_ZBA`: The Zba address generation extension is** supported, as defined in version 1.0 of the Bit-Manipulation ISA extensions.
  - **`RISCV_HWPROBE_EXT_ZBB`: The Zbb extension is supported, as defined** in version 1.0 of the Bit-Manipulation ISA extensions.
  - **`RISCV_HWPROBE_EXT_ZBS`: The Zbs extension is supported, as defined** in version 1.0 of the Bit-Manipulation ISA extensions.
- `RISCV_HWPROBE_KEY_CPUPERF_0`: A bitmask that contains performance information about the selected set of processors.
  - `RISCV_HWPROBE_MISALIGNED_UNKNOWN`: The performance of misaligned accesses is unknown.
  - `RISCV_HWPROBE_MISALIGNED_EMULATED`: Misaligned accesses are emulated via software, either in or below the kernel. These accesses are always extremely slow.
  - `RISCV_HWPROBE_MISALIGNED_SLOW`: Misaligned accesses are slower than equivalent byte accesses. Misaligned accesses may be supported directly in hardware, or trapped and emulated by software.
  - `RISCV_HWPROBE_MISALIGNED_FAST`: Misaligned accesses are faster than equivalent byte accesses.
  - `RISCV_HWPROBE_MISALIGNED_UNSUPPORTED`: Misaligned accesses are not supported at all and will generate a misaligned address fault.

CHAPTER

SIX

ARCH/RISCV MAINTENANCE GUIDELINES FOR DEVELOPERS

6.1 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.

6.2 Patchwork

RISC-V has a patchwork instance, where the status of patches can be checked:

https://patchwork.kernel.org/project/linux-riscv/list/

If your patch does not appear in the default view, the RISC-V maintainers have likely either requested changes, or expect it to be applied to another tree.

Automation runs against this patchwork instance, building/testing patches as they arrive. The automation applies patches against the current HEAD of the RISC-V *for-next* and *fixes* branches, depending on whether the patch has been detected as a fix. Failing those, it will use the RISC-V *master* branch. The exact commit to which a series has been applied will be noted on patchwork. Patches for which any of the checks fail are unlikely to be applied and in most cases will need to be resubmitted.

6.3 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 unlikely to be incompatibly changed in the future. For specifications from the RISC-V foundation this means "Frozen" or "Ratified", for the UEFI forum specifications this means a published ECR. (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 implementers 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 consider patches for extensions that either:

- Have been officially frozen or ratified by the RISC-V Foundation, or
- Have been implemented in hardware that is widely available, per standard Linux practice.

(Implementers, may, of course, maintain their own Linux kernel trees containing code for any custom extensions that they wish.)

CHAPTER

SEVEN

RISC-V LINUX USER ABI

7.1 ISA string ordering in /proc/cpuinfo

The canonical order of ISA extension names in the ISA string is defined in chapter 27 of the unprivileged specification. The specification uses vague wording, such as should, when it comes to ordering, so for our purposes the following rules apply:

1. Single-letter extensions come first, in canonical order. The canonical order is "IMAFDQLCBKJTPVH".
2. All multi-letter extensions will be separated from other extensions by an underscore.
3. Additional standard extensions (starting with 'Z') will be sorted after single-letter extensions and before any higher-privileged extensions.
4. For additional standard extensions, the first letter following the 'Z' conventionally indicates the most closely related alphabetical extension category. If multiple 'Z' extensions are named, they will be ordered first by category, in canonical order, as listed above, then alphabetically within a category.
5. Standard supervisor-level extensions (starting with 'S') will be listed after standard unprivileged extensions. If multiple supervisor-level extensions are listed, they will be ordered alphabetically.
6. Standard machine-level extensions (starting with 'Zxm') will be listed after any lower-privileged, standard extensions. If multiple machine-level extensions are listed, they will be ordered alphabetically.
7. Non-standard extensions (starting with 'X') will be listed after all standard extensions. If multiple non-standard extensions are listed, they will be ordered alphabetically.

An example string following the order is:

```
rv64imadc_zifoo_zigoo_zafoo_sbar_scar_zxmbaz_xqux_xrux
```

7.2 Misaligned accesses

Misaligned accesses are supported in userspace, but they may perform poorly.

CHAPTER

EIGHT

VECTOR EXTENSION SUPPORT FOR RISC-V LINUX

This document briefly outlines the interface provided to userspace by Linux in order to support the use of the RISC-V Vector Extension.

8.1 1. prctl() Interface

Two new prctl() calls are added to allow programs to manage the enablement status for the use of Vector in userspace. The intended usage guideline for these interfaces is to give init systems a way to modify the availability of V for processes running under its domain. Calling these interfaces is not recommended in libraries routines because libraries should not override policies configured from the parant process. Also, users must noted that these interfaces are not portable to non-Linux, nor non-RISC-V environments, so it is discourage to use in a portable code. To get the availability of V in an ELF program, please read `COMPAT_HWCAP_ISA_V` bit of `ELF_HWCAP` in the auxiliary vector.

- prctl(PR_RISCV_V_SET_CONTROL, unsigned long arg)

  Sets the Vector enablement status of the calling thread, where the control argument consists of two 2-bit enablement statuses and a bit for inheritance mode. Other threads of the calling process are unaffected.

  Enablement status is a tri-state value each occupying 2-bit of space in the control argument:

  - `PR_RISCV_V_VSTATE_CTRL_DEFAULT`: Use the system-wide default enablement status on execve(). The system-wide default setting can be controlled via sysctl interface (see sysctl section below).
  - `PR_RISCV_V_VSTATE_CTRL_ON`: Allow Vector to be run for the thread.
  - `PR_RISCV_V_VSTATE_CTRL_OFF`: Disallow Vector. Executing Vector instructions under such condition will trap and casuse the termination of the thread.

  arg: The control argument is a 5-bit value consisting of 3 parts, and accessed by 3 masks respectively.

  The 3 masks, PR_RISCV_V_VSTATE_CTRL_CUR_MASK, PR_RISCV_V_VSTATE_CTRL_NEXT_MASK, and PR_RISCV_V_VSTATE_CTRL_INHERIT represents bit[1:0], bit[3:2], and bit[4]. bit[1:0] accounts for the enablement status of current thread, and the setting at bit[3:2] takes place at next execve(). bit[4] defines the inheritance mode of the setting in bit[3:2].

- PR_RISCV_V_VSTATE_CTRL_CUR_MASK: bit[1:0]: Account for the Vector enablement status for the calling thread. The calling thread is not able to turn off Vector once it has been enabled. The prctl() call fails with EPERM if the value in this mask is PR_RISCV_V_VSTATE_CTRL_OFF but the current enablement status is not off. Setting PR_RISCV_V_VSTATE_CTRL_DEFAULT here takes no effect but to set back the original enablement status.
- PR_RISCV_V_VSTATE_CTRL_NEXT_MASK: bit[3:2]: Account for the Vector enablement setting for the calling thread at the next execve() system call. If PR_RISCV_V_VSTATE_CTRL_DEFAULT is used in this mask, then the enablement status will be decided by the system-wide enablement status when execve() happen.
- PR_RISCV_V_VSTATE_CTRL_INHERIT: bit[4]: the inheritance mode for the setting at PR_RISCV_V_VSTATE_CTRL_NEXT_MASK. If the bit is set then the following execve() will not clear the setting in both PR_RISCV_V_VSTATE_CTRL_NEXT_MASK and PR_RISCV_V_VSTATE_CTRL_INHERIT. This setting persists across changes in the system-wide default value.

Return value:

- 0 on success;
- EINVAL: Vector not supported, invalid enablement status for current or next mask;
- EPERM: Turning off Vector in PR_RISCV_V_VSTATE_CTRL_CUR_MASK if Vector was enabled for the calling thread.

On success:

- A valid setting for PR_RISCV_V_VSTATE_CTRL_CUR_MASK takes place immediately. The enablement status specified in PR_RISCV_V_VSTATE_CTRL_NEXT_MASK happens at the next execve() call, or all following execve() calls if PR_RISCV_V_VSTATE_CTRL_INHERIT bit is set.
- Every successful call overwrites a previous setting for the calling thread.

- prctl(PR_RISCV_V_GET_CONTROL)

  Gets the same Vector enablement status for the calling thread. Setting for next execve() call and the inheritance bit are all OR-ed together.

  Note that ELF programs are able to get the availability of V for itself by reading COMPAT_HWCAP_ISA_V bit of ELF_HWCAP in the auxiliary vector.

  Return value:

  - a nonnegative value on success;
  - EINVAL: Vector not supported.

8.2 2. System runtime configuration (sysctl)

To mitigate the ABI impact of expansion of the signal stack, a policy mechanism is provided to the administrators, distro maintainers, and developers to control the default Vector enablement status for userspace processes in form of sysctl knob:

- /proc/sys/abi/riscv_v_default_allow

  Writing the text representation of 0 or 1 to this file sets the default system enablement status for new starting userspace programs. Valid values are:

  - 0: Do not allow Vector code to be executed as the default for new processes.
  - 1: Allow Vector code to be executed as the default for new processes.

  Reading this file returns the current system default enablement status.

  At every execve() call, a new enablement status of the new process is set to the system default, unless:

  - PR_RISCV_V_VSTATE_CTRL_INHERIT is set for the calling process, and the setting in PR_RISCV_V_VSTATE_CTRL_NEXT_MASK is not PR_RISCV_V_VSTATE_CTRL_DEFAULT. Or,
  - The setting in PR_RISCV_V_VSTATE_CTRL_NEXT_MASK is not PR_RISCV_V_VSTATE_CTRL_DEFAULT.

  Modifying the system default enablement status does not affect the enablement status of any existing process of thread that do not make an execve() call.

8.3 3. Vector Register State Across System Calls

As indicated by version 1.0 of the V extension [1], vector registers are clobbered by system calls.

1: https://github.com/riscv/riscv-v-spec/blob/master/calling-convention.adoc

CHAPTER

NINE

FEATURE STATUS ON RISCV ARCHITECTURE

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

Table 1 – continued from

Subsystem	Feature	Kconfig	Status
vm	ELF-ASLR	ARCH_WANT_DEFAULT_TOPDOWN_MMAP_LAYOUT	ok
vm	huge-vmap	HAVE_ARCH_HUGE_VMAP	ok
vm	ioremap_prot	HAVE_IOREMAP_PROT	TODO
vm	PG_uncached	ARCH_USES_PG_UNCACHED	TODO
vm	pte_special	ARCH_HAS_PTE_SPECIAL	ok
vm	THP	HAVE_ARCH_TRANSPARENT_HUGEPAGE	ok