x86 Feature Flags

Introduction

On x86, flags appearing in /proc/cpuinfo have an X86_FEATURE definition in arch/x86/include/asm/cpufeatures.h. If the kernel cares about a feature or KVM want to expose the feature to a KVM guest, it can and should have an X86_FEATURE_* defined. These flags represent hardware features as well as software features.

If users want to know if a feature is available on a given system, they try to find the flag in /proc/cpuinfo. If a given flag is present, it means that the kernel supports it and is currently making it available. If such flag represents a hardware feature, it also means that the hardware supports it.

If the expected flag does not appear in /proc/cpuinfo, things are murkier. Users need to find out the reason why the flag is missing and find the way how to enable it, which is not always easy. There are several factors that can explain missing flags: the expected feature failed to enable, the feature is missing in hardware, platform firmware did not enable it, the feature is disabled at build or run time, an old kernel is in use, or the kernel does not support the feature and thus has not enabled it. In general, /proc/cpuinfo shows features which the kernel supports. For a full list of CPUID flags which the CPU supports, use tools/arch/x86/kcpuid.

How are feature flags created?

a: Feature flags can be derived from the contents of CPUID leaves.

These feature definitions are organized mirroring the layout of CPUID leaves and grouped in words with offsets as mapped in enum cpuid_leafs in cpufeatures.h (see arch/x86/include/asm/cpufeatures.h for details). If a feature is defined with a X86_FEATURE_<name> definition in cpufeatures.h, and if it is detected at run time, the flags will be displayed accordingly in /proc/cpuinfo. For example, the flag "avx2" comes from X86_FEATURE_AVX2 in cpufeatures.h.

b: Flags can be from scattered CPUID-based features.

Hardware features enumerated in sparsely populated CPUID leaves get software-defined values. Still, CPUID needs to be queried to determine if a given feature is present. This is done in init_scattered_cpuid_features(). For instance, X86_FEATURE_CQM_LLC is defined as 11*32 + 0 and its presence is checked at runtime in the respective CPUID leaf [EAX=f, ECX=0] bit EDX[1].

The intent of scattering CPUID leaves is to not bloat struct cpuinfo_x86.x86_capability[] unnecessarily. For instance, the CPUID leaf [EAX=7, ECX=0] has 30 features and is dense, but the CPUID leaf [EAX=7, EAX=1] has only one feature and would waste 31 bits of space in the x86_capability[] array. Since there is a struct cpuinfo_x86 for each possible CPU, the wasted memory is not trivial.

c: Flags can be created synthetically under certain conditions for hardware features.

Examples of conditions include whether certain features are present in MSR_IA32_CORE_CAPS or specific CPU models are identified. If the needed conditions are met, the features are enabled by the set_cpu_cap or setup_force_cpu_cap macros. For example, if bit 5 is set in MSR_IA32_CORE_CAPS, the feature X86_FEATURE_SPLIT_LOCK_DETECT will be enabled and "split_lock_detect" will be displayed. The flag "ring3mwait" will be displayed only when running on INTEL_FAM6_XEON_PHI_[KNL|KNM] processors.

d: Flags can represent purely software features.

These flags do not represent hardware features. Instead, they represent a software feature implemented in the kernel. For example, Kernel Page Table Isolation is purely software feature and its feature flag X86_FEATURE_PTI is also defined in cpufeatures.h.

Naming of Flags

The script arch/x86/kernel/cpu/mkcapflags.sh processes the #define X86_FEATURE_<name> from cpufeatures.h and generates the x86_cap/bug_flags[] arrays in kernel/cpu/capflags.c. The names in the resulting x86_cap/bug_flags[] are used to populate /proc/cpuinfo. The naming of flags in the x86_cap/bug_flags[] are as follows:

a: The name of the flag is from the string in X86_FEATURE_<name> by default.

By default, the flag <name> in /proc/cpuinfo is extracted from the respective X86_FEATURE_<name> in cpufeatures.h. For example, the flag "avx2" is from X86_FEATURE_AVX2.

b: The naming can be overridden.

If the comment on the line for the #define X86_FEATURE_* starts with a double-quote character ("), the string inside the double-quote characters will be the name of the flags. For example, the flag "sse4_1" comes from the comment "sse4_1" following the X86_FEATURE_XMM4_1 definition.

There are situations in which overriding the displayed name of the flag is needed. For instance, /proc/cpuinfo is a userspace interface and must remain constant. If, for some reason, the naming of X86_FEATURE_<name> changes, one shall override the new naming with the name already used in /proc/cpuinfo.

c: The naming override can be "", which means it will not appear in /proc/cpuinfo.

The feature shall be omitted from /proc/cpuinfo if it does not make sense for the feature to be exposed to userspace. For example, X86_FEATURE_ALWAYS is defined in cpufeatures.h but that flag is an internal kernel feature used in the alternative runtime patching functionality. So, its name is overridden with "". Its flag will not appear in /proc/cpuinfo.

Flags are missing when one or more of these happen

a: The hardware does not enumerate support for it.

For example, when a new kernel is running on old hardware or the feature is not enabled by boot firmware. Even if the hardware is new, there might be a problem enabling the feature at run time, the flag will not be displayed.

b: The kernel does not know about the flag.

For example, when an old kernel is running on new hardware.

c: The kernel disabled support for it at compile-time.

For example, if 5-level-paging is not enabled when building (i.e., CONFIG_X86_5LEVEL is not selected) the flag "la57" will not show up [1]. Even though the feature will still be detected via CPUID, the kernel disables it by clearing via setup_clear_cpu_cap(X86_FEATURE_LA57).

d: The feature is disabled at boot-time.

A feature can be disabled either using a command-line parameter or because it failed to be enabled. The command-line parameter clearcpuid= can be used to disable features using the feature number as defined in /arch/x86/include/asm/cpufeatures.h. For instance, User Mode Instruction Protection can be disabled using clearcpuid=514. The number 514 is calculated from #define X86_FEATURE_UMIP (16*32 + 2).

In addition, there exists a variety of custom command-line parameters that disable specific features. The list of parameters includes, but is not limited to, nofsgsbase, nosmap, and nosmep. 5-level paging can also be disabled using "no5lvl". SMAP and SMEP are disabled with the aforementioned parameters, respectively.

e: The feature was known to be non-functional.

The feature was known to be non-functional because a dependency was missing at runtime. For example, AVX flags will not show up if XSAVE feature is disabled since they depend on XSAVE feature. Another example would be broken CPUs and them missing microcode patches. Due to that, the kernel decides not to enable a feature.

[1] 5-level paging uses linear address of 57 bits.