5-level paging

Overview

Original x86-64 was limited by 4-level paging to 256 TiB of virtual address space and 64 TiB of physical address space. We are already bumping into this limit: some vendors offer servers with 64 TiB of memory today.

To overcome the limitation upcoming hardware will introduce support for 5-level paging. It is a straight-forward extension of the current page table structure adding one more layer of translation.

It bumps the limits to 128 PiB of virtual address space and 4 PiB of physical address space. This "ought to be enough for anybody" Â©.

QEMU 2.9 and later support 5-level paging.

Virtual memory layout for 5-level paging is described in Documentation/x86/x86_64/mm.rst

Enabling 5-level paging

CONFIG_X86_5LEVEL=y enables the feature.

Kernel with CONFIG_X86_5LEVEL=y still able to boot on 4-level hardware. In this case additional page table level -- p4d -- will be folded at runtime.

User-space and large virtual address space

On x86, 5-level paging enables 56-bit userspace virtual address space. Not all user space is ready to handle wide addresses. It's known that at least some JIT compilers use higher bits in pointers to encode their information. It collides with valid pointers with 5-level paging and leads to crashes.

To mitigate this, we are not going to allocate virtual address space above 47-bit by default.

But userspace can ask for allocation from full address space by specifying hint address (with or without MAP_FIXED) above 47-bits.

If hint address set above 47-bit, but MAP_FIXED is not specified, we try to look for unmapped area by specified address. If it's already occupied, we look for unmapped area in *full* address space, rather than from 47-bit window.

A high hint address would only affect the allocation in question, but not any future mmap()s.

Specifying high hint address on older kernel or on machine without 5-level paging support is safe. The hint will be ignored and kernel will fall back to allocation from 47-bit address space.

This approach helps to easily make application's memory allocator aware about large address space without manually tracking allocated virtual address space.

One important case we need to handle here is interaction with MPX. MPX (without MAWA extension) cannot handle addresses above 47-bit, so we need to make sure that MPX cannot be enabled we already have VMA above the boundary and forbid creating such VMAs once MPX is enabled.