Lenovo

Introduction to 5-Level Paging in 3rd Gen Intel Xeon Scalable Processors with Linux

Introduces the background of page tables with x86_64 platform

Describes the structure of multilevel paging, both 4-level and 5-level paging

Shows how to enable and disable 5-level paging using kernel parameter

Provides the method for verification of the 5-level paging feature

Huaisheng Ye

LENOVO PRESS

Abstract

5-level paging is a new paging mode of x86_64 processors in EM64T mode. As its name suggests, 5-level paging will translate linear addresses by traversing a 5-level hierarchy of paging structures. Compared to the traditional 4-level paging, 5-level paging extends the processor's linear-address width to 57 bits from 48 bits, thereby giving the processor and applications access to an even larger amount of memory.

This paper introduces the background of 5-level paging function from both the hardware and software perspective. It also explains how to enable or disable this function in the Linux kernel and how to verify that the function is working properly.

This paper is for administrators and application programmers. An understanding of x86 multilevel paging mechanism and basic rules of memory management is useful for the underlying implementation of this feature.

At Lenovo® Press, we bring together experts to produce technical publications around topics of importance to you, providing information and best practices for using Lenovo products and solutions to solve IT challenges.

See a list of our most recent publications at the Lenovo Press web site:

http://lenovopress.com

Do you have the latest version? We update our papers from time to time, so check whether you have the latest version of this document by clicking the **Check for Updates** button on the front page of the PDF. Pressing this button will take you to a web page that will tell you if you are reading the latest version of the document and give you a link to the latest if needed. While you're there, you can also sign up to get notified via email whenever we make an update.

Contents

Introduction

Linux uses address-translation support called *paging*, which translates linear addresses (virtual addresses) which are used by OS, to physical addresses which are used to access memory (or memory-mapped I/O).

Page tables are used to map virtual address space to physical address space. Most areas of virtual address spaces are not used, and most of them are not associated with page frames. As a result, a far less memory intensive model that fulfills the same purpose can be used, known as *multilevel paging*.

To reduce the size of page tables and to allow unneeded areas to be ignored, the x86_64 architecture split each virtual address into five parts. The first four parts are used as page tables and the last part is used as offset in every page. That is what we call as *4-level paging*.

On x86 processors supporting 64-bit architecture, modern operating systems like Linux typically references memory using linear addressing, which requires that the processor be configured to use paging mechanism that translates linear addresses to physical addresses. The processor uses the resulting physical addresses to access memory.

Page tables map a virtual memory address to the physical address where the data is actually stored. It is conceptually a linear array indexed by the virtual address (or, at least, by the page-frame-number portion of that address) and yielding the page-frame number of the associated physical memory page. However, such an array would be very large a waste of memory resources. Most user applications don't use the full available virtual address space even on 32-bit systems, and they just use a tiny fraction of it on 64-bit systems, so the address space tends to be sparsely populated and, as a result, much of that array would go unused.

The alternative to using a linear array is to instead implement a sparse tree representing the address space. x86 processors have implemented such addressing for the past decade. The design is shown in Figure 1.

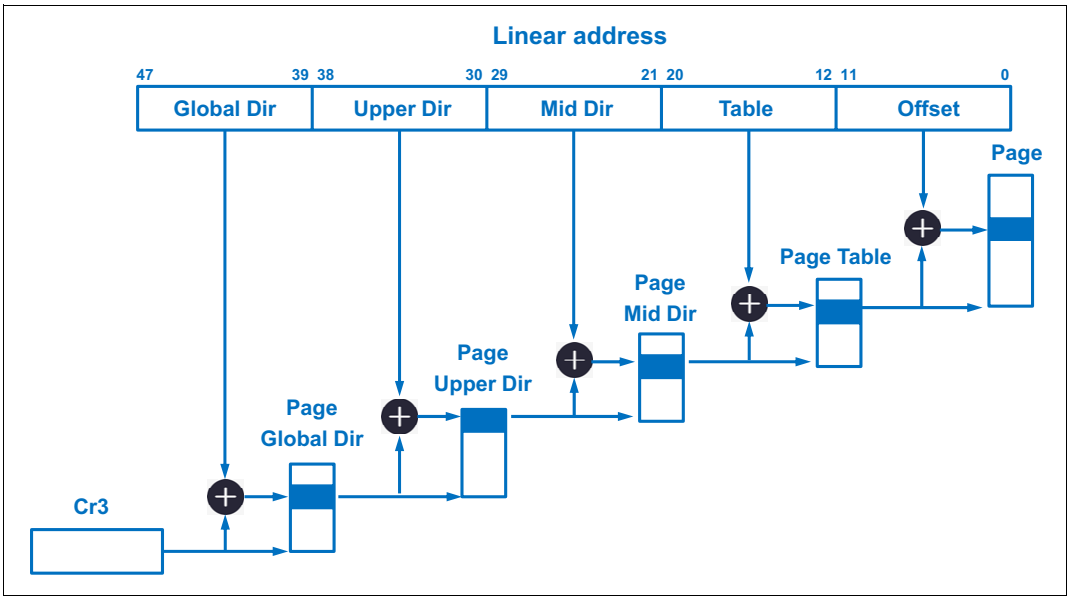

Figure 1 Structure of the 4-level paging

The top row of boxes in Figure 1 represents the bits of a 64-bit virtual address. To translate that address, the CPU hardware splits the address into several bit fields. The scheme shows here how the x86_64 architecture utilizes those bit fields.

1. The uppermost 16 bits are discarded, only the lower 48 bits of the virtual address are used.
2. Of the bits that are used, the nine most significant (bits 39-47) are used to index into the page global directory (PGD).
3. PUD represents a single page for each address space. The value read here is the address of the page upper directory (PUD).
4. Bits 30-38 of the virtual address are used to index into the indicated PUD page to get the address of the page middle directory (PMD).
5. With bits 21-29, the PMD can be indexed to get the lowest level page table, just called the PTE.
6. Finally, bits 12-20 of the virtual address will, when used to index into the PTE, yield the physical address of the actual page containing the data. The lowest twelve bits of the virtual address are the offset into the memory page itself.

When 5-level paging had been merged to Linux kernel upstream, there are some concerns about application adaptability and performance regression.

Although 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, Linux kernel is 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. For more information, see the following page:

https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/x86/x86_64/5level-paging.rst

There is concern of performance regression when 5-level paging has been merged to kernel upstream. The performance concerns were both for 5-level code on 4-level hardware and 5-level paging on 5-level hardware when the greater extended physical/virtual address space isn't needed. But it has been addressed already, mainstream Linux distributions enabled this feature and also upstream kernel v5.5 enabled it by default as well. For more information, see the following page:

https://www.phoronix.com/scan.php?page=news_item&px=Linux-5.5-5-Level-Paging

Increasing the memory limit with 5-level paging

5-level paging is an extension that would relax that limit to 57 linear address bits. In order to support 57 linear address, the Linux kernel expands the capability of MM (Memory Management) subsystem to 5-level paging.

The 5-level paging table is extended by inserting P4 Dir in bits 39-47 in the existing 4-level paging table. Using 5-level paging increases the virtual address space from 256 TiB to 128 PiB and the physical address space from 64 TiB to 4 PiB. Intel's 5-level paging works by extending the size of virtual addresses from 48 to 57 bits, and the physical addresses from 46 to 52 bits.

Table 1 shows the address range difference and merged window between the 4-level paging and 5-level paging mechanisms.

In ThinkSystem™ V2 servers, 5-level paging will work with 3rd Gen Intel Xeon Scalable processor family (codename Ice Lake) and beyond. AMD EPYC 7002 and 7003 processors do not support 5-level paging.

Table 1 Comparison of 4/5-level paging

Range	4-level paging	5-level paging (Ice Lake)
Virtual Address	256 TiB = 2^{48}	128 PiB = 2^{57}
Userspace	128 TiB	64 PiB
Physical Address	64 TiB = 2^{46}	4 PiB = 2^{52}
Merged to upstream	v2.6.10	v4.11-rc2

Figure 2 shows how 5-level paging works with 3rd Gen Intel Xeon Scalable (Ice Lake) processors. There are another 9 bits (48 - 56) can be used as the highest linear address.

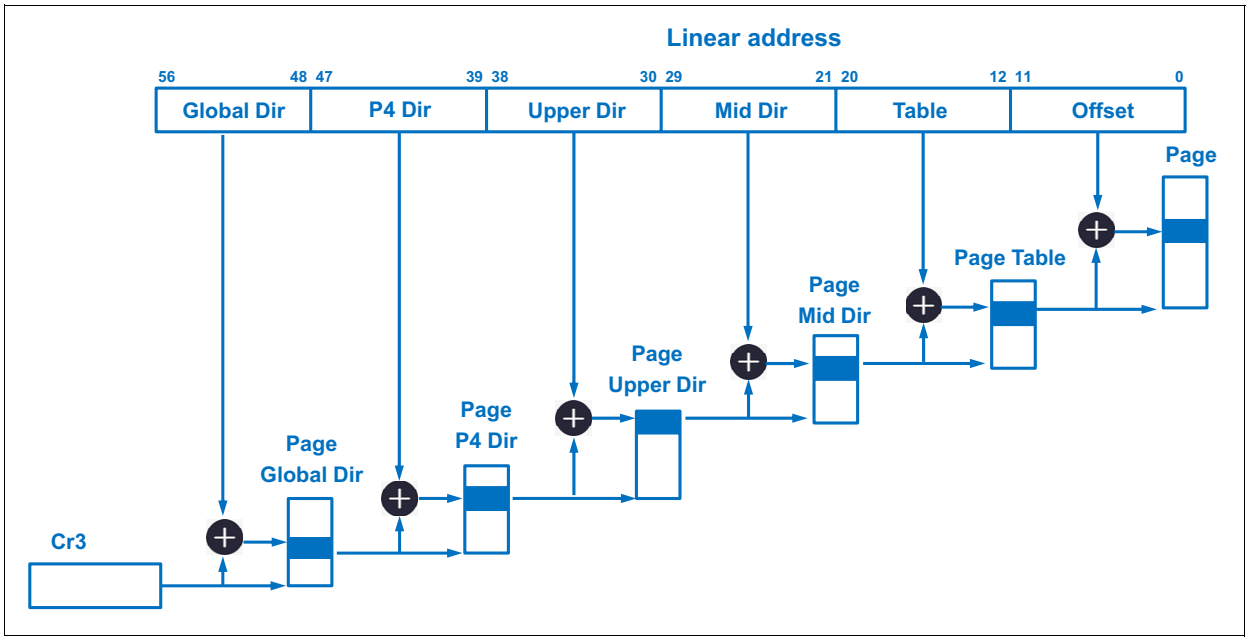

Figure 2 Structure of the 5-level paging

Enabling and verifying 5-level paging in Linux

5-level paging is currently only supported in 3rd Gen Intel Xeon Scalable processors (code name Ice Lake). 5-level paging also requires Linux kernel support. This section describes how to verify the prerequisite kernel support and how to switch between 4-level and 5-level paging.

Enabling 5-level paging

For Linux distributions, 5-level paging is enabled automatically if the processor supports the functionality and if the kernel has the capability.

Windows and VMware ESXi currently do not support 5-level paging.

Verify support in the Linux kernel package

5-level paging is supported only in certain Linux distributions and only with certain kernel versions. Table 2 shows the status of enablement for 5-level paging with tier 1 Linux distributions.

Table 2 Support for 4-level and 5-level paging in Linux

Tier 1 Linux distributions	4-level paging	5-level paging
RHEL 7	Supported	Not supported
RHEL 8 and later	Supported	Supported
SLES 12	Supported	Not supported
SLES 15	Supported	Not supported
SLES 15 SP1	Supported	Not supported
SLES 15 SP2 and later	Supported	Supported
Ubuntu	Supported	Not supported

Figure 3 shows the command to check the config file of RHEL 8.1 GA to check whether 5-level paging can be enabled in the kernel:

```
$ cat /boot/config-$(uname -r) | grep -E 'X86_5LEVEL|PGTABLE_LEVELS'
CONFIG_PGTABLE_LEVELS=5
CONFIG_X86_5LEVEL=y
```

Figure 3 Checking if 5-level paging can be enabled in RHEL 8.1

From the output above, it shows that 5-level paging (X86_5LEVEL) has been enabled with current kernel, and the levels of page table (PGTABLE_LEVELS) is 5.

Figure 4 shows the command to check the config file of SLES 15 SP1 default kernel (kernel base is 4.12.14):

```
$ cat /boot/config-$(uname -r) | grep -E 'X86_5LEVEL|PGTABLE_LEVELS'
CONFIG_PGTABLE_LEVELS=4
```

Figure 4 Checking if 5-level paging can be enabled in SLES 15 SP1

This output shows that the kernel only supports a 4-level page table.

On some earlier versions of the kernel, no output is shown because the kernel configuration file doesn't have any related items such as CONFIG_ X86_5LEVEL and CONFIG_PGTABLE_LEVELS.

Figure 5 shows the output of RHEL 7.8 whose kernel code base is 3.10.

```
$ cat /boot/config-$(uname -r) | grep -E 'X86_5LEVEL|PGTABLE_LEVELS'
```

Figure 5 Checking if 5-level paging can be enabled in RHEL 7.8

Verify support in the operating system

Although the kernel package and processor support 5-level paging, you will also need to check whether 5-level paging has been enabled successfully by the OS.

A simple way for verifying support is by using the command in Figure 6.

```
$ lscpu | grep -i la57
Flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm
constant_tsc art arch_perfmon … avx512_vnni avx512_bitalg tme avx512_vpopcntdq
la57 rdpid md_clear pconfig flush_l1d arch_capabilities
```

Figure 6 Verifying support in the OS for 5-level paging

The presence of flag `la57` (highlighted in red in Figure 6) indicates that the platform supports 5-level paging feature.

Disabling 5-level paging

There may be scenarios where the server hardware and Linux kernel support the 5-level paging mechanism however you do not wish to have it enabled.

To disable 5-level paging, make changes to grub as follows. Figure 7 shows how to apply the change to RHEL 8 for example.

```
$ vim /etc/default/grub
# Append no5lvl to GRUB_CMDLINE_LINUX
GRUB_CMDLINE_LINUX="crashkernel=auto
resume=UUID=226afee7-87b9-42fe-99ff-a755e24a2da7 rhgb quiet no5lvl"

# If RHEL 8 is installed with Legacy mode,
$ grub2-mkconfig -o /boot/grub2/grub.cfg

# Verify parameters have been appended successfully,
$ cat /boot/grub2/grubenv
# GRUB Environment Block
saved_entry=b4cf75ab98c04fd2a933a0df751fda95-4.18.0-193.el8.x86_64
kernelopts=root=UUID=1ce3d82f-62f3-4156-8265-480279f345ba ro crashkernel=auto
resume=UUID=226afee7-87b9-42fe-99ff-a755e24a2da7 rhgb quiet no5lvl
boot_success=0

# If RHEL 8 is installed with UEFI mode, use this command below instead.
$ grub2-mkconfig -o /boot/efi/EFI/centos/grub.cfg
```

Figure 7 Parameter to disable 5-level paging

You will need to reboot the server for these changes to take effect.

Test case for 5-level and 4-level paging

This section describes the test case that has been published with Linux kernel tool to let users verify 5-level and 4-level paging.

Verify userspace address by mmap using 5-level paging

Here is an example with RHEL 8 when 5-level paging works. The steps are as follows:

1. Load UEFI setup to default and install OS for testing
2. Boot Linux and log in it with root account
3. Make sure your test server can download file from Internet by proxy if plan to use wget
4. Download va_128TBswitch.c from kernel.org or download from another server and use scp to copy the file locally:

   # **wget https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/plain/tools/testing/selftests/vm/va_128TBswitch.c**

5. Compile the source file:

   # **gcc -o va_128 va_128TBswitch.c**

6. Run the program and review the output:

```
# ./va_128
mmap(ADDR_SWITCH_HINT - PAGE_SIZE, PAGE_SIZE): 0x7fc138455000 - OK
mmap(ADDR_SWITCH_HINT - PAGE_SIZE, (2 * PAGE_SIZE)): 0x7fc138454000 - OK
mmap(ADDR_SWITCH_HINT, PAGE_SIZE): 0x800000000000 - OK
mmap(ADDR_SWITCH_HINT, 2 * PAGE_SIZE, MAP_FIXED): 0x800000000000 - OK
mmap(NULL): 0x7fc138454000 - OK
mmap(LOW_ADDR): 0x40000000 - OK
mmap(HIGH_ADDR): 0x1000000000000 - OK
mmap(HIGH_ADDR) again: 0xffffc138457000 - OK
mmap(HIGH_ADDR, MAP_FIXED): 0x1000000000000 - OK
mmap(-1): 0xffffc138455000 - OK
mmap(-1) again: 0xffffc138453000 - OK
mmap(ADDR_SWITCH_HINT - PAGE_SIZE, PAGE_SIZE): 0x7fc138455000 - OK
mmap(ADDR_SWITCH_HINT - PAGE_SIZE, 2 * PAGE_SIZE): 0x7fc138454000 - OK
mmap(ADDR_SWITCH_HINT - PAGE_SIZE/2 , 2 * PAGE_SIZE): 0x7fc138452000 - OK
mmap(ADDR_SWITCH_HINT, PAGE_SIZE): 0x800000000000 - OK
mmap(ADDR_SWITCH_HINT, 2 * PAGE_SIZE, MAP_FIXED): 0x800000000000 - OK
```

The expected result is to see OK at the end of each line, indicating that all tests are successful and that 5-level paging is working correctly.

Verify userspace address by mmap using 4-level paging

As mentioned in “Disabling 5-level paging” on page 7, if you want to use 4-level paging with a kernel that is enabled for 5-level paging, use the no5lvl parameter to disable 5-level paging. The steps are as follows:

1. Add `no5lvl` to kernel parameters as described in “Disabling 5-level paging” on page 7.
2. Reboot the server and make sure `no5lvl` can be found by either of the following:

   # **cat /proc/cmdline**
   # **lscpu**

3. Run va_128 and review the output:

```
# ./va_128
mmap(ADDR_SWITCH_HINT - PAGE_SIZE, PAGE_SIZE): 0x7fb2f762d000 - OK
mmap(ADDR_SWITCH_HINT - PAGE_SIZE, (2 * PAGE_SIZE)): 0x7fb2f762c000 - OK
mmap(ADDR_SWITCH_HINT, PAGE_SIZE): 0x7fb2f762d000 - OK
mmap(ADDR_SWITCH_HINT, 2 * PAGE_SIZE, MAP_FIXED): 0xffffffffffffffff - FAILED
mmap(NULL): 0x7fb2f762b000 - OK
mmap(LOW_ADDR): 0x40000000 - OK
mmap(HIGH_ADDR): 0x7fb2f762b000 - OK
mmap(HIGH_ADDR) again: 0x7fb2f7629000 - OK
mmap(HIGH_ADDR, MAP_FIXED): 0xffffffffffffffff - FAILED
mmap(-1): 0x7fb2f7627000 - OK
mmap(-1) again: 0x7fb2f7625000 - OK
mmap(ADDR_SWITCH_HINT - PAGE_SIZE, PAGE_SIZE): 0x7fb2f7626000 - OK
mmap(ADDR_SWITCH_HINT - PAGE_SIZE, 2 * PAGE_SIZE): 0x7fb2f7625000 - OK
mmap(ADDR_SWITCH_HINT - PAGE_SIZE/2 , 2 * PAGE_SIZE): 0x7fb2f7623000 - OK
mmap(ADDR_SWITCH_HINT, PAGE_SIZE): 0x7fb2f7622000 - OK
mmap(ADDR_SWITCH_HINT, 2 * PAGE_SIZE, MAP_FIXED): 0xffffffffffffffff - FAILED
```

The expected result is that the mmap request with MAP_FIXED flags will fail with `MAP_FAILED (0xffffffffffffffff)`.

Summary

Linux Kernel developers developed 5-level paging to overcome the x86_64 limitation of 256 TiB of virtual address space and 64 TiB of physical address space. If user have that much memory in single server node, that limitation is being hit but can be increased through 5-level paging.

There are only two conditions that have to be met if user want to enable this feature. The first is processor and server platform hardware need to support 5-level paging in hardware level. The other is Linux Kernel should enable 5-level paging in the software level.

References

For more information, see these resources:

- Article “Five-level page tables” by Jonathan Corbet

  https://lwn.net/Articles/717293/

- Intel paper, *5-Level Paging and 5-Level EPT*

  https://software.intel.com/content/dam/develop/public/us/en/documents/5-level-paging-white-paper.pdf

- Article “Linux 5.5 To Enable Intel's 5-Level Paging Support By Default” by Michael Larabel

  https://www.phoronix.com/scan.php?page=news_item&px=Linux-5.5-5-Level-Paging

Author

Huaisheng Ye is a Linux kernel engineer in the Lenovo Infrastructure Solutions Group, based in Beijing, China and is the on-site Lenovo engineer at Red Hat. He is dedicated to feature enablement and troubleshooting for Linux distributions and also contributes patches to the kernel upstream. He has eleven years' experience with the Linux kernel in the power, memory management, storage and NVDIMM subsystems. Currently he focuses on projects for optimizing the storage infrastructure of software-defined storage (SDS), based on modern cache implementation.

Special thanks to the following people for their contributions and suggestions for this project:

- Kelvin Shieh, Advisory Engineer, OS and Preload DDPM development
- Adrian Huang, Linux Kernel Engineer, OS and Preload
- Liupeng, Advisory Engineer, OS and Preload

Notices

Lenovo may not offer the products, services, or features discussed in this document in all countries. Consult your local Lenovo representative for information on the products and services currently available in your area. Any reference to a Lenovo product, program, or service is not intended to state or imply that only that Lenovo product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any Lenovo intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any other product, program, or service.

Lenovo may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

Lenovo (United States), Inc.
1009 Think Place - Building One
Morrisville, NC 27560
U.S.A.
Attention: Lenovo Director of Licensing

LENOVO PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. Lenovo may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

The products described in this document are not intended for use in implantation or other life support applications where malfunction may result in injury or death to persons. The information contained in this document does not affect or change Lenovo product specifications or warranties. Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of Lenovo or third parties. All information contained in this document was obtained in specific environments and is presented as an illustration. The result obtained in other operating environments may vary.

Lenovo may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any references in this publication to non-Lenovo Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this Lenovo product, and use of those Web sites is at your own risk.

Any performance data contained herein was determined in a controlled environment. Therefore, the result obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

This document was created or updated on May 23, 2021.

Send us your comments via the **Rate & Provide Feedback** form found at
http://lenovopress.com/lp1468

Trademarks

Lenovo and the Lenovo logo are trademarks or registered trademarks of Lenovo in the United States, other countries, or both. These and other Lenovo trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by Lenovo at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of Lenovo trademarks is available from https://www.lenovo.com/us/en/legal/copytrade/.

The following terms are trademarks of Lenovo in the United States, other countries, or both:

Lenovo® Lenovo(logo)® ThinkSystem™

The following terms are trademarks of other companies:

Intel, Xeon, and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.