Virtual Memory Management in Modern Linux Systems

Whitepaper

SUSE
A NOVELL COMPANY

Virtual Memory Management

in Modern Linux Systems – Facts and Myths

This document deals with the basics of virtual memory management on systems with an Intel architecture, with the goal of deriving fundamental conclusions for the implementation of two alternative systems in the Linux kernel. A second section of the paper justifies the decision for implementation at SUSE LINUX AG. By way of providing a corollary, we then explain simple rules regarding application profiles for which differences in virtual memory management are generally important, followed by several comparative notes on 64-bit platforms.

The purpose of this whitepaper is to help further understanding of the type of virtual memory management used in a Linux kernel from SUSE LINUX AG. An enhancement currently offered by other Linux OS vendors under the name "4/4 GB Split" for kernel implementation is not required in a SUSE kernel.

Virtual Memory Management in Modern Linux Systems

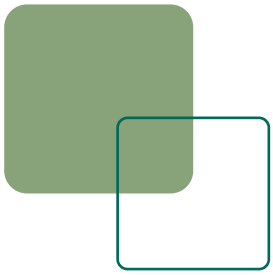
Basics

First, some background information and a few fundamental facts that describe
(a) why virtual memory management is necessary in the first place, and
(b) other issues that are relevant in this context.

As long as there have been computers, programmers have been struggling with the fact that there are different types of memory they can use for their programs and data. In addition, different storage media vary substantially with respect to their data delivery speed or general data processing speed.

Simply put, there are two types of storage devices whose processing speeds differ significantly, which is why they are used for different purposes.
(1) Main memory (RAM and caches)
(2) External mass storage devices

Main memory refers to the actual portion of memory accessible to a program. If one program, or the combination of several applications and the operating system require more memory than is *physically* available, the system is reaching its limits. In these situations, Linux provides so-called *virtual memory*, which makes it possible for programs to run on systems with significantly less *physical storage capacity* than is actually needed to run all programs at the same time. All types of main memory are temporary, which means that their contents are lost after, for example, a power failure. External mass storage devices are primarily used for storing data intended to survive such emergency situations. In addition, they are also used to provide *virtual memory*.



A 32-bit CPU can manage a maximum of 4 GB (4,294,967,296 bytes) at a time. This also means that an application program itself, for example, cannot become larger than 4 GB. Such a program, however, usually does not run autonomously. Instead, it communicates with other programs and in doing so, uses operating system functions, which in Linux are provided primarily by the Linux kernel. This means that a certain segment of the 4 GB of memory (normally, just under 1 GB) is required by the kernel in order to enable the application program to communicate. The Linux kernel uses this segment to create a table, among other things, that represents the system's actual memory configuration. In the process, the system organizes available memory into so-called storage pages of 4 KB each.

If the user now wishes to quasi-simultaneously run multiple programs that all require less than 4 GB of memory each, but together require a total of significantly more than 4 GB of memory, he faces a dilemma. This dilemma was recognized early on and it was solved by *providing virtual memory*. In the process, the operating system simulates nearly infinite storage capacity that can be made available for use by the application. The system accomplishes this by swapping out storage pages that are currently not in use from the faster main memory to the slower external mass storage device. The operating system uses the gaps that become available in this manner to provide the requesting program with actual memory capacity. The amount of memory that can be made available is limited in practice only by the size of the external mass storage device. Such procedures are known as *"swapping"* and *"paging"*. *Swapping* refers to a full swapping out of memory segments, while *paging* primarily refers to hardware support provided by the *memory management unit* (MMU), which normally assists the CPU or is integrated in it. When the system accesses memory pages that are currently hidden, the CPU receives a signal that enables the operating system to show the page again, for example, in exchange for another page that has not been in use for some time.

The Dilemma

However, we do not want to create the impression that virtual memory management can stretch physical limitations. The fact remains that a 32-bit CPU cannot directly address more than 4 GB of memory, and therefore a program cannot become larger than 4 GB, minus the amount of data described above needed to manage a program. However, the issue of *data needed by the operating system to manage a program* is precisely a factor that should not be underestimated, as illustrated below.

Linux can administer up to a maximum of 64 GB of physical memory on Intel-compatible 32-bit systems. Since no more than 4 GB of data is visible or usable at a time, the system uses additional memory in excess of 4 GB through a feature developed by Intel, the so-called *physical address extension* (PAE). In Linux, this type of memory is referred to as *high memory*, in contrast to permanent *low memory* that can be addressed directly. PAE is a 3-step process that, simply put, takes storage capacity from *high memory* and temporarily inserts it in the visible 4 GB segment.

The Linux kernel lets the user configure the ratio of the application program's usable/reserved memory to the memory reserved for the kernel, so that it is possible to realize a maximum size of 3.5 GB for the application program. Since the usual ratio is 3:1, it is apparent that maximum memory usage can be tight for certain parts of the kernel (e. g., the table in which the kernel manages all of the memory pages in the system). Therefore, in specific individual cases the decision to use a ratio of 3.5:0.5 makes sense. For each memory page that Linux manages in the system, the kernel allocates a descriptor that takes up 44 bytes of space.

It isn't difficult to calculate the size of this table alone: ((n x bytes RAM) / 4,096 bytes) * 44 bytes

One fact should not be overlooked as part of the larger picture, *PAE is a slower process than direct addressing of "only" a maximum of 4 GB of storage capacity*. Tests have shown that extensive use of PAE results in an *average decrease of 5% in overall system performance*. Moreover, if a program itself also uses additional options, such as the temporary inclusion of additional memory via *tmpfs*, one should expect a performance drop of at least 10%. Therefore, the operating system's, or rather the virtual memory manager's actual memory management efficiency is a very important factor. However, PAE and the specified measures for increasing a system's memory itself still work faster than an operating system that must actually swap memory in and out of the main memory to and from external, slow hard drives.

In the development of Linux, there have been at least two types of memory management algorithms that have evolved since the introduction of kernel version 2.4. The original algorithm was developed by *Rik van Riel* (who is currently a kernel developer at Red Hat) and soon turned out to be inadequate for the serious usage of Linux on productive systems. To solve this problem, *Andrea Arcangeli* (kernel developer at SUSE LINUX AG) developed a new type of virtual memory management, which is generally known today as *AA-VM*. By now, it is commonly used as the basis and standard for 2.4 kernels. Van Riel further developed his version and created the so-called *RMAP-VM*. This version, however, has some fundamental restrictions compared to the *AA-VM* standard.

The Dilemma

The most fundamental functional difference between *RMAP-VM* compared to *AA-VM* may well be its usage of kernel memory in general and the additional usage of administration data (*RMAP chains*) in *low memory*. Compared to *AA-VM*, this means that the system must make use of memory from the *high memory* pool much more quickly, which impairs overall performance by an additional 10%, on average.

In the final analysis, *RMAP-VM* can create a situation where an application program operating under extreme conditions has less memory available than it would with *AA-VM*. To resolve this situation, Red Hat recently announced a new feature that

(a) is intended to create a 4/4 GB split between kernel and application program, and

(b) moves the Red Hat kernel farther away from the standard.

Another very general observation is that this solution clearly slows down each instance of access from the kernel to the user area. Each time the system switches modes like this it must cycle through all of the page tables in order to take an address that is valid in the kernel and convert it into a corresponding address that is valid in the user area. It is easy to imagine what this means, for example, in terms of the resulting bandwidth for copy actions via a gigabit interface while, for example, a database application is working simultaneously to insert a new record.

The purpose of the 4/4 GB split is to make available 4 GB of memory to the application program as well as to the Linux kernel. The above explanations show that

- data segments must be reserved for communication between both parts, so that less than 4 GB of memory is actually available to the application,
- each interrupt that occurs during the processing of the application program and the kernel necessarily requires switching of those memory segments located outside of the current process context, which further impairs performance, and
- communication between the two segments is significantly slower, since it is not possible to access the user area directly from the kernel.

Consequently, with *RMAP-VM*, the 4/4 split avoids limitation of the maximum memory available for use by the application program. These limitations are caused by the allocation in *low memory* of portions of administration information (RMAP chains) that the Linux kernel uses to address virtual memory and swapped out memory, if applicable. However, significant consequences remain for system performance at a magnitude of at least 10-15% performance loss, on average, compared to a standard kernel. Due to the characteristics of SUSE LINUX kernels and the *AA-VM* they contain, a 4/4 GB split makes sense – if it does at all – only for systems with at least 32 GB of main memory or more. Such systems can generate memory mapping-related administration overhead of a magnitude that can truly limit an application in terms of available resources to the point that 3 GB of memory are no longer available to the application.

possible solutions

The facts listed here illustrate that

(a) a current SUSE LINUX kernel with *AA-VM* has sufficient capacity to run application programs on systems with a maximum of up to 32 GB of main memory – without even coming close to exhausting system resources, as is the case with *RMAP-VM*, and

(b) use of *high memory* on a 32-bit Intel system has noticeable consequences for system performance as a whole.

If you are truly faced with the need to provide an application program with more than 4 GB of usable address space, we recommend that you switch to a system with a 64-bit architecture and address space. In reality, you should already consider doing this with programs that need more than 2-3 GB of memory for a single process.

64-bit architectures were already marketed a decade ago to avoid the restrictions imposed by 32-bit addressing. Initially, these architectures were marketed for use with large enterprise applications. Given today's standard applications, however, a mid-sized company already pretty much runs up against the performance limitations of today's 32-bit platforms. With the introduction of 64-bit platforms, Intel and AMD are now reaching markets that were unavailable just a few years ago. Due to their appealing prices, it will be a long time before 32-bit platforms play a less significant role in the market than they do today. This fact, along with the huge selection of 32-bit software, are among the reasons why systems based on current Intel IA-32 architecture will need a 32-bit emulation mode in the foreseeable future. The SUSE LINUX implementation provides native support for both modes.

By now, there are a number of systems specifically designed for use in the scenarios described here. These systems are now as stable and as well tested as 32-bit systems. Moreover, AMD's 64-bit processors are a cost-efficient variant that not only provide the luxury of *549,755,813,888 bytes* of directly addressable memory but also a *high-performance 32-bit compatibility mode* that allows users to continue using all their long-time familiar, stable and well-tested software without any changes. (Today, Linux supports a maximum of 512 GB of address space per application program, while today's hardware already provides an additional 2^{48} bits of virtual addressing). Intel itself, of course, has already been offering 64-bit processors for quite some time (Itanium 2, IA-64). IBM/Motorola's PPC970 provides another option when IA-32 compatibility is not required, as does IBM's zSeries, which should also be included in these considerations. Finally, it should be kept in mind that the above issues do not affect the average user, but only those 3% of all users, who, for example, are already using SAP today with accordingly giant database applications.

Virtual memory management is an issue of vital importance wherever applications are running up close to or even beyond the limits of actual memory capacity. Current examples are database systems or SAP application servers for which the 32-bit capacity of real, addressable memory is no longer sufficient. Applications that do not fall in this category generally do not derive any advantages from 64-bit systems with shifted physical boundaries. Examples of this application category include word processing programs, small to medium-sized databases and web servers.

At the time of this document's creation, SUSE estimates that the issues discussed above are of relevance to about 3% of all newly purchased systems.