**Set 1 Question 1:** Please explain the Linux Memory Management (post 2.6) in details. What are memory zones and how do they help?

**Answer 1:**The 2.6 Linux kernel employs a large number of techniques to improve the use of large amount of memory, making Linux more enterprise-ready than ever before. Techniques used by Linux kernel 2.6 are listed below:

**(i)Reverse mapping:** In the Linux memory manager, page tables keep track of the physical pages of memory that are used by a process and page tables map virtual pages to physical pages. It is possible that some of the pages might not be used for a long period of time, so they are good choices to swap out and bring other pages into main memory for better utilization of memory. But before swapping out a page, page table of every process which is using that page needs to be updated. For this we need to check every process which is using that page. If the number of processes in the system increases, the work required to update these page tables also increases because we need to update page table of every process.

Reverse mapping provides a mechanism for discovering which processes are using a given physical page of memory. Reverse mapping does not traverse the page tables for every process. For each physical page, a linked list containing pointers to the page-table entries (PTEs) of every process currently mapping that page.This linked list is called a PTE chain*.* The PTE increases the speed of memory manager to find those processes which are using a particular page.

Even though reverse mapping provides this advantage to increase efficiency and speed of the processor but it incurs a memory overhead. Some memory has to be used to keep track of all those reverse mappings. Each entry in the PTE chain uses 4 bytes to store a pointer to the page-table entry and an additional 4 bytes to store the pointer to the next entry on the chain.

Reverse mapping has the following advantages:

* We can unmap a page from all processes which are using it without searching the virtual memory of that process.
* We can reduce the number of minor page faults by unmapping only those pages we really want to evict without scanning the virtual memory of all processes and unmapping more pages than we want to evict.

**(ii) Huge Pages:** In a typical x86 system, the memory manager deals with page size of 4KB. Consider a process whose size is 1 GB and memory manager stores it with page size of 4KB, total number of page table entries required will be 262,144.Now, consider the size of page table entry is 8 bytes. So the total overhead for 1 GB of memory mapped is 2MB. If the number of processes is large, a lot of memory is wasted on overhead which might exceed the amount of memory the application requested for use. One way to overcome this problem is that we can use a larger page size. Nearly all modern processors support at least 2 different page sizes, a small and a large one, and some processors support even more than 2 different page sizes. On typical x86 processor, the large page size is 4 MB, or 2MB on systems with physical address extension turned on. Considering the same example as we discussed above with larger page size of 4 MB. Now the process of 1 GB requires just 256 page table entries which was 262,144 in case of 4 KB page size. So the overhead with larger page size of 4KB is 2048 bytes. The overhead for 1 GB process is reduced drastically from 2 MB to 2048 bytes by using a large page size.

The number of translation lookaside buffer (TLB) misses can also be reduced by using a larger page size. Hence, improving the performance of translation lookaside buffer. The TLB is a sort of cache for the page tables that allows virtual to physical address translation to be performed more quickly for pages that are listed in the table. Using large pages, we can accommodate more memory using fewer pages. So we can reference more memory with larger page size compared to small page size.

**(iii) Storing page-table entries in high memory:** - On 32-bit machines, page tables are stored only in low memory. This low memory is limited to the first 896 MB of physical memory. And this low memory is also required for the use of rest of the kernel as well. Low memory can quickly become scarce if the number of processes in the system is high. In the Linux kernel 2.6 onwards, a configuration option called *Highmem PTE* available. It allows the page-table entries to be placed in high memory, freeing more of the low memory area which can be used for other kernel data structures. But the only problem is that processes storing their page tables in high memory are somewhat slower than the processes whose page tables are stored in low memory. However, if a system has a large number of processes are running, then it is better to store page tables in high memory so the lower memory will always remain available for kernel.

**(iv)Kernel Shared Memory:** One more change introduced in the 2.6.32 kernel is Kernel Shared Memory (KSM).KSM allows the hypervisor to increase the number of concurrent virtual machines by consolidating identical memory pages. This is known as *Kernel Samepage Merging.*KSM isuseful in virtualized as well as non-virtualized environments. KSM is also beneficial in embedded Linux systems, indicating the flexibility of the KSM.

KSM exists as a daemon in the kernel (called ksmd). KSMD scans to identify duplicate pages and merges duplicates to free pages for other uses. And this process of scanning, identifying and merging of duplicate pages are performed by ksmd periodically. It is done by ksmd in such a way that it is transparent to the user. For example, duplicate pages are merged and later on one of the users of the page make any changes to the page, that user will receive his or her own copy (in a Copy on Write fashion).

KSM searches for identical pages once it is enabled. When KSM finds an identical page, it keeps one page in a write-protected Copy On Write fashion and free up the duplicate page so that space can be used for other purposes. In the KSM, two red-black trees are used to manage the pages .One of the red-black tree is temporary. The first tree is called the *unstable tree* which is used to store new pages that are not considered as stable. Pages that are unchanged for some period of time are stored in unstable tree. Pages which are stored in this unstable tree are not write protected. The second tree is called the *stable tree* which is used tostore the pages that have been found to be stable and merged by KSM. KSM uses a simple 32-bit checksum, to identify whether a page is volatile or non-volatile. When a page is scanned, KSM calculates its checksum and this checksum is stored within the same page. On a subsequent scan, we compute the checksum again, if the newly computed checksum is different from the previously generated checksum, the page is modified and is therefore not a good candidate for merging.

**(v) Stability:** Better stability is another important improvement of the Linux Kernel 2.6 memory manager. In previous kernels, users started having memory management-related stability problems.

Memory management was one of the most scrutinized areas of kernel development in 2.6. The new memory management code has been tested and optimized on everything from very low end desktop systems to large, enterprise-class, multi-processor systems.

[[1]](#endnote-2)

1. http://www.ijetch.org/papers/115--T258.pdf [↑](#endnote-ref-2)