CMPSCI 577 Operating Systems Design and Implementation **Spring 2020**

Lecture 21: 04/14

Lecturer: ***Prashant Shenoy*** *Scribe:* ***Shashwat***

21.1 Memory management in LINUX

This lecture discusses about Kernel Memory allocations, user-space memory management and caching in memory.

21.2 Kernel Management

The allocation done by the kernel differs from the user space allocation.

Physcial pages are treated by the kernel as the basic unit of memory management

21.3 Pages in Linux

Earlier pages existed with default size of 4KB whereas now Huge pages exist in addition to the 4KB pages.

64 bit Linux allows upto 128 TB of virtual address space and can address 64 TB of physical memory.

While assigning memory to a process is relatively cheap memory management is not since every memory access requires a page walk and typically takes around 10 to 100 CPU cycles.

21.4 Large Memory Applications

These include applications having size more than 10GB

Having large page sizes enables a significant reduction in memory walks for such an application.

21.5 Huge pages

It is possible to map 2M and 1G pages using entries in the second and third level page tables, such pages in LINUX are known as **Huge**

Uses of such kinds of pages significantly reduces the pressure on TLB, improves TLB hit rate and thus improves overall system performance.

One disadvantage of these is that they cause fragmentation.

21.6 Transparent huge pages

This is based on the concept of Huge pages and is the default for all applications today and is transparent to the application

The idea is to assign a huge page to applications requiring large memory chunk

In many cases THB actually causes severe application issues in many applications

21.7 Memory in the kernel

Every physical page is represented by the kernel through a **struct** page structure

The page structure is associated with physical pages and not virtual pages and the total size of the structure can exceed 100 MB

21.8 Page Flags

These flags indicate whether a page is dirty or locked in memory. Atleast 32 flags are simultaneously available.

21.9 Tracking Pages

The _count stores the usage count of the page which basically indicates how many references exist to this page.

Possible owners of the page includes user-space processes, dynamically allocated kernel data, static kernel code, page cache and so on.

21.10 Memory Zones

The kernel cannot treat all pages as identical because of hardware limitations.

The kernel divides pages into different zones, in which the pages with similar properties are grouped together.

21.11 The main zones

The use and layout of memory zones is architecture dependent and not all zones need to exist. Each zone is represented by struct zone in the mmzone.h file.

21.12 Freeing pages in the kernel

It must be ensured that only the page which are allocated should be freed. Passing the wrong page struct or address or the incorrect order can result in corruption. The Kernel unlike the user space trusts itself.

21.13 Allocating byte sized chunks

kmalloc Used if physically contiguous memory is needed which is mostly the case for hardware devices.

vmalloc Used when virtually contiguous memory is needed.

21.14 Linux slab layer

The slab layer acts as a generic data structure-caching layer and also like an internal caching layer in the kernel. The slab layer can enable more control to the kernel.

21.15 Slab layer basic tenets

- Frequently used data structures tend to be allocated and freed frequently so they are cached.
- Frequent allocation and deallocation can lead to memory fragmentation.
- If allocators are aware of concepts such as object_size, page size and total cache size, it can make more intelligent decisions.
- If part of the cache is made per processor allocation and frees can be performed without an SMP lock
- If the allocator is NUMA-aware, it can fulfill allocations from the same memory node as the requestor
- Stored objects can be colored to prevent multiple objects from mapping to the same cache lines

21.16 Slab design

- Different objects are divided into groups called **caches**, these caches are divided into **slabs** which are composed of one or more physically **contiguous pages**
- There is 1 cache per object type which is used for process descriptors
- Each slab contains some number of objects which are the data structures being cached and each slab is in one of the three states, full, partial or empty.

21.17 Kernel Stack

- Kernel processes have non dynamic stacks in which the stack per process is one memory page.

21.18 The Process Address Space

21.18.1 Address space layout

- Determined (mostly) by the application
- Determined at compile time
- OS usually reserves part of the address space to map itself
- Application can dynamically request new mappings from the OS, or delete mappings

21.18.2 The process memory areas

Memory areas contain, for example

- A memory map of the executable file's code, called the text section.
- A memory map of the executable file's initialized global variables, called the data section.
- A memory map of the zero page containing uninitialized global variables (called bss section)
- A memory map of the zero page used for the process's user-space stack
- An additional text, data, and bss section for each shared library, such as the C library and dynamic linker, loaded into the process's address space
- Any memory mapped files.
- Any shared memory segments.
- Any anonymous memory mappings, such as those associated with malloc()

21.18.3 Virtual Memory Areas

Linux represents portions of a process with a vm_area_struct, or vma

21.18.4 The Memory Descriptor

- The kernel represents a process's address space with a data structure called the memory descriptor. It contains all the information related to the process address space and is represented by struct mm_struct.
- The memory descriptor associated with a given task is stored in the mm field of the task's process descriptor.

21.19 The page cache and page writeback

21.19.1 Page cache

- RAM can be orders of magnitude faster than disk

- The page cache consists of physical pages in RAM, the contents of which correspond to physical blocks on a disk.
- Entire files need not be cached
- The page cache can hold some files in their entirety while storing only a page or two of other files. What is cached depends on what has been accessed

21.19.2 Write Caching

- It belongs to one of the 3 categories, no-write, Write-through-cache and Write-back
- The write-back caching policy requires that a write operation occurs at the cache only
- Write can be performed in bulk, optimizing access to the slow disk
- Application can force immediate write back with sync system calls (and some open/mmap options)

21.19.3 The Linux Page Cache

- A page in the page cache can consist of multiple non-contiguous physical disk blocks
- The kernel must check for the existence of a page in the page cache before initiating any page I/O

21.19.4 Cache reclamation

- Kernel caches and processes can continue assigning memory until memory becomes scarce
- Memory pages can be divided into one of four categories: Unreclaimable, Swappable, Syncable and Discardable

21.19.5 Cache eviction policies

- Least Recently Used
- The two list strategy
  - Linux keeps two lists: the active list and the inactive list.
  - Pages on the active list are considered "hot" and are not available for eviction.
  - Pages on the inactive list are available for cache eviction
  - Pages are placed on the active list only when they are accessed while already residing on the inactive list
  - The lists are kept in balance
  - Approach is also known as LRU/2; it can be generalized to n-lists, called LRU/n