**LINUX MEMORY MANAGEMENT**

-Sai Sirisha Nadiminti (PG1, 2020201044)

Linux memory management has two components[[1]](#endnote-2). First is the management of physical memory that deals with allocating and freeing physical memory. Then comes the handling of virtual memory. The virtual memory is memory-mapped into the address spaces of the running processes. The Linux virtual memory system is responsible for maintaining the address space accessible to each process. It creates pages of virtual memory on demand and manages loading those pages from disk and swapping them back out to disk as required.

This report will focus on the management of physical memory using zones, buddy system and slab allocation in Linux.

**Management of physical memory using zones**

Linux separates physical memory into four different zones, or regions that are architecture specific:

* ZONE\_DMA - In the Intel x86-32 architecture, certain ISA (industry standard architecture) devices can only access the lower 16 MB of physical memory using DMA. On these systems, the first 16 MB of physical memory comprise ZONE\_DMA.
* ZONE\_DMA32 - certain devices can only access the first 4 GB of physical memory, despite supporting 64-bit addresses. On such systems, the first 4 GB of physical memory comprise ZONE\_DMA32.
* ZONE\_HIGHMEM – ZONE\_HIGHMEM (high memory) refers to physical memory that is not mapped into the kernel address space. For example, on the 32-bit Intel architecture (where 232 provides a 4-GB address space), the kernel is mapped into the first 896 MB of the address space; the remaining memory is referred to as high memory and is allocated from ZONE\_HIGHMEM.
* ZONE NORMAL – This zone comprises everything else - the normal, regularly mapped pages.

Whether an architecture has a given zone depends on its hardware constraints. For example, a modern, 64-bit architecture such as Intel x86-64 has a small 16 MB ZONE DMA for legacy devices and all the rest of its memory in ZONE\_NORMAL, with no high memory.

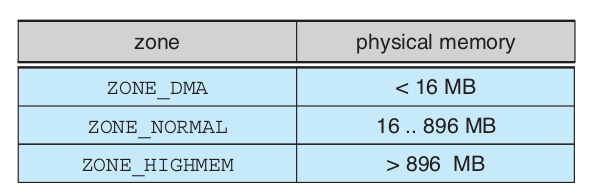


Fig 1. The relationship of zones and physical addresses on the Intel x86-32 architecture

**Need of zones[[2]](#endnote-3):**

The kernel cannot treat all pages as identical due to hardware limitations. Some pages, because of their physical address in memory, cannot be used for certain tasks. Due to this limitation, the kernel divides pages into different zones. For example, we have the **ZONE\_DMA**, which is from 0 - 16MB. This is needed for older ISA Devices, which are not capable of adressing above the 16MB limit. Then we have the **ZONE\_NORMAL**, where most of the kernel operations take place and is adressed permanently into the kernels adress space. The 1GB and 3GB split is simple. We have virtual adresses here, so for our application, the memory adress always starts at 0x00000000, reserved are the 1st GB of this for kernel stuff. In kernel mode you are allowed to use system calls. If we would not have the kernel memory mapped to the virtual adress space, we would have to do a context switch to trap into kernel mode, which is time consuming. But as kernel-mode operations can take place in the same virtual adress space, context switch is no longer needed to allocate new memory or do any other system call.

The kernel maintains a list of free pages for each zone. When a request for physical memory arrives, the kernel satisfies the request using the appropriate zone. The primary physical-memory manager in the Linux kernel is the page allocator. Each zone has its own allocator, which is responsible for allocating and freeing all physical pages for the zone and is capable of allocating ranges of physically contiguous pages on request. The allocator uses a buddy system to keep track of available physical pages. It takes less time to allocate and deallocate pages and also minimizes external fragmentation.[[3]](#endnote-4)

**Buddy system**

In this scheme, adjacent units of allocatable memory are paired together. Each allocatable memory region has an adjacent partner (or buddy). Whenever two allocated partner regions are freed up, they are combined to form a larger region - a buddy heap. That larger region also has a partner with which it can combine to form a still larger free region. Conversely, if a small memory request cannot be satisfied by allocation of an existing small free region, then a larger free region will be subdivided into two partners to satisfy the request. Separate linked lists are used to record the free memory regions of each allowable size. Under Linux, the smallest size allocatable under this mechanism is a single physical page.

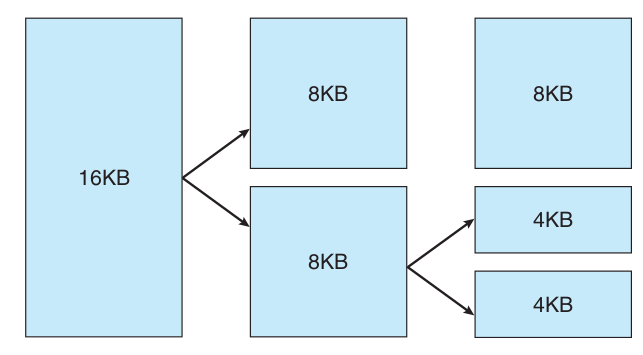


Fig 2. Splitting of memory in the buddy system

In Fig 2., we can see that a 4 KB region is being allocated, but the smallest available region is 16 KB . The region is broken up recursively until a piece of the desired size is available.

**Slab allocation**

Slab system works upon buddy algorithm for allocating memory to kernel data structures. A slab is made up of one or more physically contiguous pages. A cache consists of one or more slabs. There is a single cache for each unique kernel data structure - for example, a cache for the data structure representing process descriptors, a cache for file objects, a cache for inodes, etc.

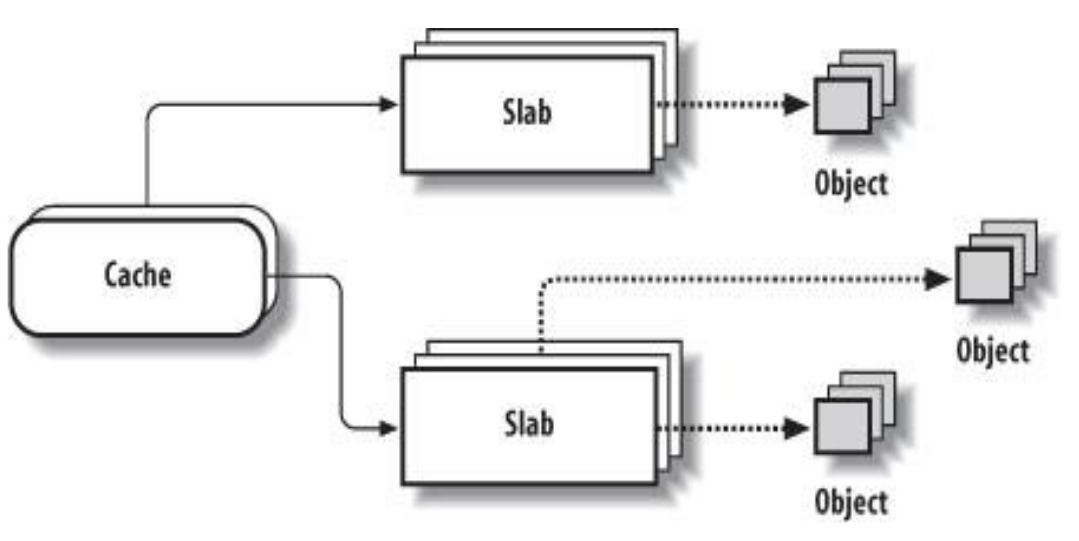


Fig 3. A cache can contain one or more slabs

In Linux, a slab may be in one of three possible states:

1. Full - All objects in the slab are marked as used.

2. Empty - All objects in the slab are marked as free.

3. Partial - The slab consists of both used and free objects.

The slab allocator first attempts to satisfy the request with a free object in a partial slab. If none exist, a free object is assigned from an empty slab. If no empty slabs are available, a new slab is allocated from contiguous physical pages and assigned to a cache; memory for the object is allocated from this slab.

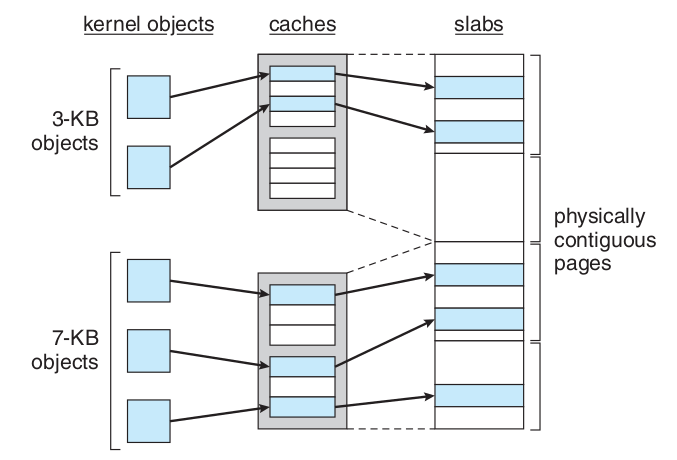


Fig 4. Slab allocator in Linux

1. Operating System Concepts, Ninth Edition by [Avi Silberschatz,](http://www.cs.yale.edu/homes/avi) [Peter Baer Galvin,](http://www.galvin.info/) [Greg Gagne](http://people.westminstercollege.edu/faculty/ggagne) [↑](#endnote-ref-2)
2. https://stackoverflow.com/questions/30566766/why-do-we-have-memory-zones-in-linux [↑](#endnote-ref-3)
3. https://www.it.iitb.ac.in/frg/wiki/images/f/fc/113050076\_Rajesh\_Prodduturi\_week5\_presentation\_3\_2012-08-04.pdf [↑](#endnote-ref-4)