Zenn
Log in

Linux memory allocation algorithm: Slab Allocation

Published on 12/24/2023 2025/03/07

Linux kernel algorithm OS memory Tech

Junya
Follow
Love Low-Level Programming

Give a badge
What is a badge giver?

table of contents

Introduction

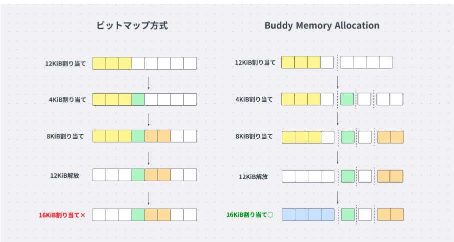
In the previous article, we introduced **Buddy Memory Allocation , one of the main memory allocation algorithms in Linux, in detail. In this article, we will focus on Slab Allocation** , another important memory allocation algorithm used in Linux , and explain how it works.

Premise

Review of the first part

In the first part, we introduced the basic principles and mechanisms of Buddy Memory Allocation.

This algorithm allocated and freed memory blocks in powers of two, reducing the occurrence of external fragmentation while speeding up the allocation and freeing process.



Comparison of Bitmap Method and Buddy Memory Allocation

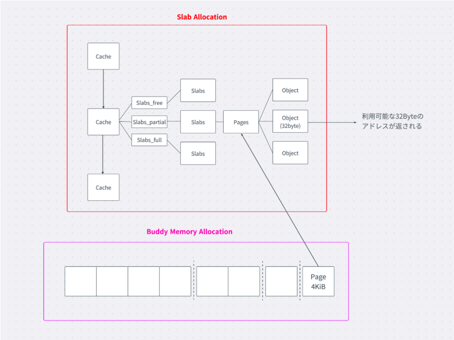
In particular, compared to contiguous memory allocation methods such as bitmaps, Buddy Memory Allocation can significantly reduce external fragmentation.
However, there are concerns that internal fragmentation may occur.

reference

News from Zenn

Registration for the 2nd AI Agent Hackathon with Google Cloud begins on A…

This time's scope



Allocating 32 bytes of memory

In this article, we will focus on Slab Allocation, the algorithm used in the Linux kernel, how it works, its benefits and concerns, and its implementation in Linux.

Throughout this two-part series, we will also aim to provide a holistic understanding of the memory allocation process depicted in the diagram above.

Slab Allocation

history

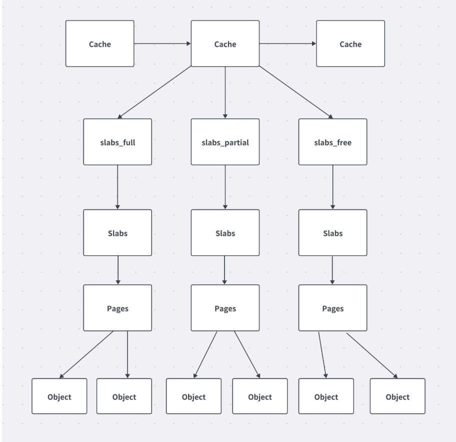
Slab Allocation is a memory allocation algorithm developed by Jeff Bonwick in 1994 and first introduced in the Solaris 2.4 kernel. It has been used in the Linux kernel since version 2.2 and continues to be used as an important memory allocation algorithm to this day. The introduction of Slab Allocation in the Linux kernel has significantly improved the efficiency of memory allocation and release.

By the way, Slab Allocation is also used in Horizon, the microkernel for **the Nintendo Switch , and Memcached , an in-memory data store, so it is an algorithm that feels familiar to us.**

overview

In a nutshell, Slab Allocation is a memory management algorithm intended to reduce internal fragmentation and improve memory allocation and freeing performance.

The diagram below shows the overall structure of Slab Allocation.



Overall picture of slab allocation

Slab Allocation is composed of elements such as **Cache** and **Slab** , and uses these to achieve high-speed memory allocation and release. We will introduce each component and follow the overall flow.

Cache

A cache is a structure for managing slabs (memory) of objects of a certain type or size. Certain objects (e.g. file and process structures) are frequently used in the kernel. Such objects of a certain size are managed by their own cache.
The slab allocation system consists of a circular list of caches called the cache chain, and when allocating memory, the appropriate cache is selected from this list.

Slab

A slab is a contiguous memory area for storing objects of a certain size.
It manages the allocated memory blocks (pages) allocated by the Buddy Memory Allocation, and keeps track of how many objects can be stored in it and the allocation status of the objects.

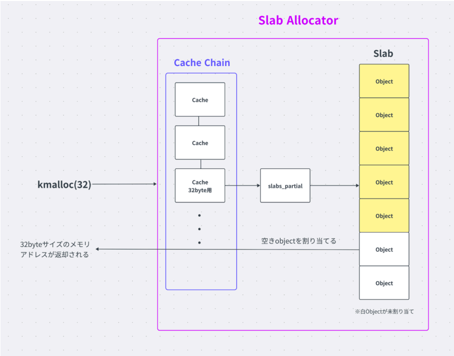
Cache manages slabs in three lists:

- **slabs_free** : Slabs that do not have any objects allocated to them are stored
- **slabs_partial** : Slabs to which some objects are allocated are stored
- **slabs_full** : Slabs with all objects allocated are stored

When allocating slabs, priority will be given to the slabs stored in slabs_partial.

Taking this information into account, the overall flow of allocation and freeing is shown below.

allocation



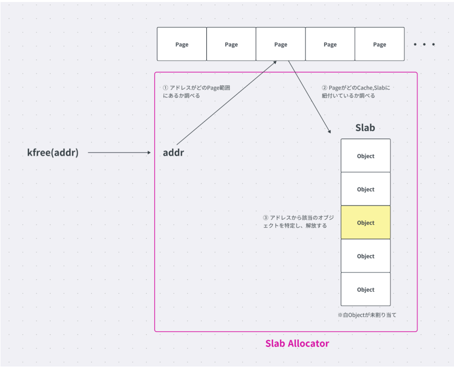
Memory Allocation in Slab Allocation

① When a request to allocate 32 bytes of memory occurs, the Allocator searches the Cache Chain for a cache of the corresponding size. If a cache of the corresponding size does not exist, a new cache of 32 bytes is created.

② Once a suitable Cache is obtained, the Allocator checks the Cache's slabs_partial list. If slabs_partial is empty, it takes a slab from slabs_free and moves it to slabs_partial.

③ Find a free object location in the selected Slab and return its memory address to the allocation request.

release



Memory release in Slab Allocation

1. When a request to release 32 bytes of memory occurs, the page range in which the memory address to be returned is included is calculated.

② Check the cache and slab associated with that page.

③ Find out which object the address is from the slab and release that object. If all objects in the slab become unallocated after this release, move the slab to slabs_free.

In this way, Slab Allocation achieves memory management using Cache and Slab.
Let's see what benefits this management method brings.

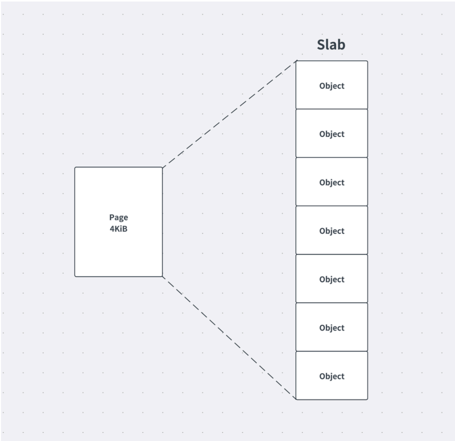
advantage

Slab Allocation provides the following benefits:

Reduces internal fragmentation

In Linux, the minimum memory block is set to 4KiB, so even if you request a memory allocation of 32 bytes, a 4KiB block of memory is actually allocated. This results in more than 3KiB of memory remaining unused, causing internal fragmentation.

Slab Allocation can reduce this internal fragmentation
by efficiently dividing the 4KiB memory blocks (pages) allocated by Buddy Memory Allocation into slabs for allocation to small objects.



Simplified diagram of Page and Slab

As shown in the above diagram, slabs can divide pages into smaller parts and allocate memory to multiple

small objects.
For example, a slab that manages a 32-byte object can theoretically manage 128 objects (although this is not the case in actual slabs, as they have a slightly more complex structure).
Using slabs in this way can reduce unused space in large memory blocks and significantly reduce internal fragmentation.

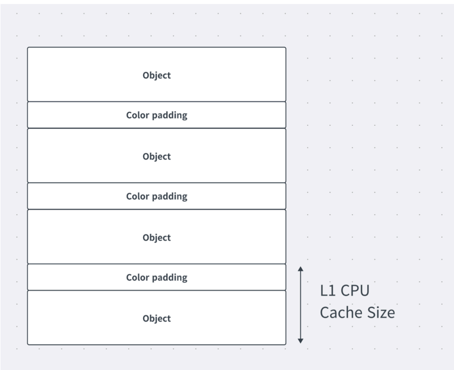
Reduces allocation and deallocation overhead

As mentioned earlier, it is common for identical objects (structures) to be used frequently in the kernel, and the process of finding free space in physical memory every time these objects are needed and releasing them when they are done with them can be costly and affect performance.

Slab Allocation reduces the overhead associated with such frequent memory allocation and freeing. Specifically, slabs with pre-allocated pages are cached in slabs_free. When a memory allocation request occurs, the Slab Allocation system immediately finds an available area among the slabs in slabs_free or slabs_partial and returns the address. This process has a cost of O(1), allowing for quick memory allocation. In this way, by preparing and caching objects in advance using Slab Allocation, the memory allocation and freeing process is accelerated, improving the overall performance of the kernel.

Hardware cache can be effectively utilized

Slab Allocation is designed to make effective use of cache memory.



Slab Structure

Slab Allocation uses a concept called coloring, which adds a slight offset between objects in a slab to improve the efficiency of cache memory usage. Cache memory holds main memory data in fixed-length units called cache lines. However, if adjacent objects are placed in the same cache line, access to one object may evict the other object from the cache, resulting in performance degradation (cache miss).

Coloring attempts to ensure that objects within the same slab are placed in different cache lines by adding an offset between the objects, reducing the chances that objects will evict each other from their cache lines, improving cache hit rates and ultimately improving overall system performance.

Concerns

While there are many benefits, there are also some concerns.

Memory usage issues

Slab Allocation pre-allocates and reserves memory for certain types of objects, which can result in inefficient memory usage as the memory reserved for less frequently used objects may be underutilized.

Management Complexity

Slab Allocation has multiple layers such as cache, slab, and object, and to manage them efficiently, complex logic is required. This complicates the kernel memory management code, which directly leads to difficulties in debugging and maintenance.

Scalability issues

Some implementations of Slab Allocation have issues with scalability on multi-core processor systems. This is because lock contention occurs when multiple processors access slabs simultaneously. Multi-core processors were not yet widespread when Slab Allocation was created, so it is not surprising that this problem occurs.

Linux implementation

The Linux kernel introduces the following three memory management methods based on the concept of Slab Allocation.

- **SLAB** : The original implementation of Slab Allocation. Based on an older architecture, it was used in early Linux kernels.
- **SLUB** : Introduced as a simpler and more powerful alternative to SLAB. SLUB is designed to be suitable for multi-core processors and focuses on reducing overhead.
- **SLOB** : A lightweight implementation of Slab Allocation designed primarily for embedded systems. Useful in resource-limited environments.

[PATCH v2 00/21] remove the SLAB…
lore.kernel.org

However, as written in this Linux kernel mailing, SLAB has

been deprecated in Linux kernel 6.5, and the code is planned to be removed for the release of Linux kernel 6.8.
SLAB is based on an architecture before multi-core processors became widespread, so it has not been fully adapted to the requirements of modern operating systems. As a result, it seems that the more efficient and simpler SLUB is moving in the direction of becoming mainstream.
The removal of SLAB can be seen as part of an effort to simplify the Linux kernel's memory management code and improve compatibility with modern hardware.
In this introduction, I explained SLAB, the original concept of Slab Allocation, so if I feel like it, I'll write an article explaining the details of SLUB in the new year.

Conclusion

In these two articles, I have written about the mechanism and role of "Buddy Memory

Allocation" and "Slab Allocation" in Linux physical memory management.

By combining these two algorithms, Linux can effectively prevent both external and internal fragmentation, achieving efficient and fast memory allocation.

The algorithm continues to evolve with the times and advances in technology, such as the transition from SLAB to SLUB. However, the fundamental idea has remained unchanged for decades and is widely adopted in many systems, so it is an algorithm worth learning.

reference

Slab Allocator
www.kernel.org

USENIX 2001 Annual Technical Conferenc…
static.usenix.org

https://events.static.linuxfound.org/sites/…
events.static.linuxfound.org

[PATCH v2 00/21] remove the SLAB…
lore.kernel.org

Slab allocation - Wikipedia
en.wikipedia.org

12

Junya

Love Low-Level Programming

Follow

Support authors by giving them badges

Authors who receive a badge will receive cash or Amazon gift cards from Zenn.

Give a badge

Discussion

Please log in to comment

Login

Read next

Alert - Hackthebox Writeup
r1o 22 hours ago ♡1

[shell/bash] When you forget how to use `` or $() in variable...
yuckey 9 hours ago ♡1

neo-c version 5.0.3 released
ab25cq 2022/02/05 ♡1

.bash_profile .bashrc scp yay
nice 1 day ago ♡1

How to register a private CA in Linux
Taka... in Mitsubishi UFJ I...
3 days ago ♡4

Ubuntu - Assigning shortcut keys to mouse buttons
Tamura,N 1 day ago ♡2

Introducing Pundit to Rails for permission management (2)
Tada 1 day ago ♡1

Introducing Pundit to Rails for permission management (1)
Tada 1 day ago

Build your own large-scale language model! (Transformers+DeepS...
celery 2023/12/13 ♡113

Detailed explanation V4L2 (video for linux 2)
yutyan in Tech Blog - Turing
2023/02/10 ♡90

Information sharing community for engineers

About Zenn
Operating company
Announcements and Releases
event

How to use
Corporate Menu New
Publication / Pro
FAQ

X (Twitter)
GitHub
Media Kit

terms of service
Privacy Policy
Special Commercial Law Notice

classmethod