Kernel Virtual Memory Allocation

Overview

Allocating Kernel Memory

- When a process running in user mode requests additional memory, pages are allocated from the list of free page frames maintained by the kernel.
- Kernel memory is often allocated form a free-memory pool different from the list used to satisfy ordinary user-mode processes.

Abbiamo detto precedentemente che non dobbiamo solo allocare memoria per lo user space, ma anche memoria per il kernel.
Dobbiamo vedere i metodi usati per allocare memoria al kernel.

Allocating Kernel Memory

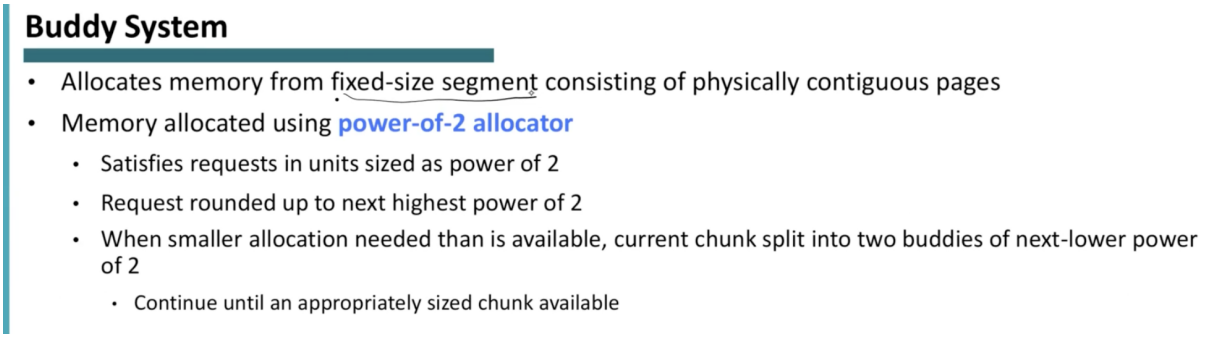
- Kernel Memory is often allocated from a free-memory pool for two reasons:
  - The kernel requests memory for data structures of varying sizes, some of which are less than a page in size. So, the kernel must minimize the waste due to fragmentation. This is important because many operating systems do not subject kernel code or data to the paging system.
  - Some kernel memory needs to be contiguous. Certain HW interacts directly with physical memory, without the benefits of a virtual memory interface.
- Two strategies for allocating Kernel Memory:
  - **Buddy System**
  - **Slab Allocation**

Di solito l'allocazione di memoria per il kernel viene fatta in modo **contiguo**.
Dobbiamo ricordarci che di solito sappiamo già da prima la struttura dati del kernel, è qualcosa di predefinito, sappiamo già di quanta memoria ha bisogno e che tipo di strutture dati deve allocare.

Quindi allochiamo memoria tramite una struttura dati predefinita.
Di solito vengono usati 2 metodi per allocare memoria kernel:

- **Buddy System**
- **Slab Allocation**

Buddy System

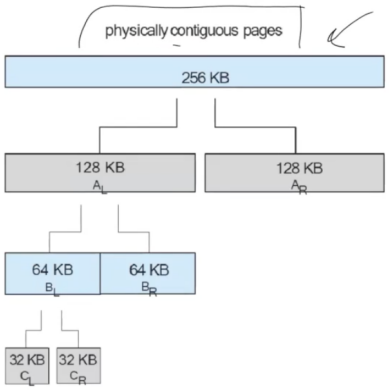


L'idea è di avere ***fixed-size segments*** di memoria (dimensione fissa) e allocazione di pagine contigua.
Un'altra caratteristica del buddy system è che si basa su potenze di 2.

Es:

Buddy System

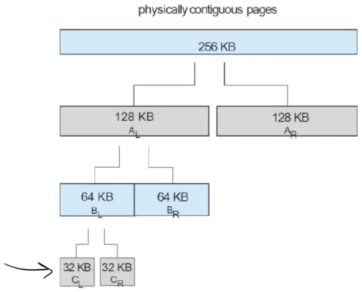
- For example, assuming the size of a memory segment is 256KB, kernel requests 21KB
  - The segment is initially divided into two **buddies** >> A_L and A_R, each 128 KB in size
  - One of these buddies is further divided into two 64KB buddies >> B_L and B_R
  - Either BL or BR is divided again into two 32 KB buddies >> C_L and C_R
    - One of these buddies is used to satisfy the 21KB request.



Immaginiamo di avere uno spazio contiguo di pagine in memoria fisica di 256KB.
Il kernel richiede 21KB di memoria.
Buddy System divide la memoria in 2 iterativamente, fino ad arrivare a ottenere un pezzetto di memoria di 32KB (potenza di 2 più vicina a contenere 21KB).

Buddy System

- An advantage of the buddy system is how quickly adjacent buddies can be combined to form larger segments using a technique known as **Coalescing**.
  - In the previous example, when the kernel release C_L unit, the system can coalesce C_L and C_R into a 64 KB segment.
  - This segment, B_L, can be coalesced with its buddy, B_R, to form a 128 KB segment.
  - Ultimately, we can end up with the original 256 KB segment.

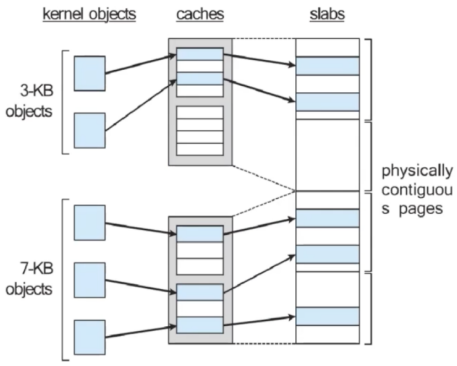


Se il kernel avrà poi bisogno di più memoria, può fare il **merge** dei blocchi di memoria creati per arrivare ad esempio a 64KB, o 128KB etc..
Questa tecnica si chiama
Coalescing.

Slab Allocation

Slab Allocation

- Another strategy for allocating kernel memory is **slab allocation**.
  - **Slab** is made up of one or more physically contiguous pages.
  - **Cache** consists of one or more slabs.
- Single cache for each unique kernel data structure
  - Each cache filled with **objects** – instantiations of the kernel data structure the cache represents.



Usato ad esempio su Linux.
Strategia che
"fa gruppi dei diversi blocchi della memoria, e poi crea un pool di questi blocchi che il kernel può poi usare".
Sappiamo che la memoria che il kernel chiede è qualcosa di predefinito: sappiamo ad esempio come è definita e quanta memoria richiede un
Thread Control Block (di solito 3KB), oppure ***Process Control Block*** (7KB).
L'idea quindi è suddividere la memoria in blocchi da 3KB o 7KB, formare quindi dei gruppi di blocchi di 3KB o 7KB, e posizionare tali gruppi in una zona di memoria che chiamiamo
cache (non è la cache dei processori che conosciamo, è una zona di memoria che identifichiamo come "*pool di gruppi di blocchi di dimensione fissa*").

Quando il kernel chiede memoria per queste varie strutture dati, va a prendere blocchi che sono disponibili in questo pool di memoria.
Abbiamo gruppi di memoria da 3KB o da 7KB.

Quando il kernel chiede di avere memoria, ad esempio per un TCB da 3KB, gli viene comunque riservato un gruppo di questi blocchi (quindi, come in slide, invece di dargli 3KB di spazio, gli diamo un gruppo di 4 spazi da 3KB ciascuno), e li va a "fillare" con i suoi oggetti finchè non esaurisce i blocchi del gruppo.

Quando finiscono i blocchi di un gruppo, alla richiesta di memoria successiva richiederà un altro gruppo di blocchi, e così via..
Dobbiamo ricordare che in questa tecnica non è possibile avere dei “singoli” blocchi da 3KB in cache, ma sempre e solo gruppi di dimensione fissa che contengono blocchi da 3KB.

Slab Allocation

- When aa cache is created, a number of objects, which are initially marked as **free**, are allocated to the cache.
- When a new object for a kernel data structure is needed, the allocator can assign any free object from the cache to satisfy the request.
  - The object assigned from the cache is marked as **used**.
- If slab is full of used objects, next object allocated from empty slab
  - If no empty slabs, new slab allocated from contiguous physical pages and assign to cache.
- Benefits include no fragmentation, fast memory request satisfaction