Paginazione a livelli multipli

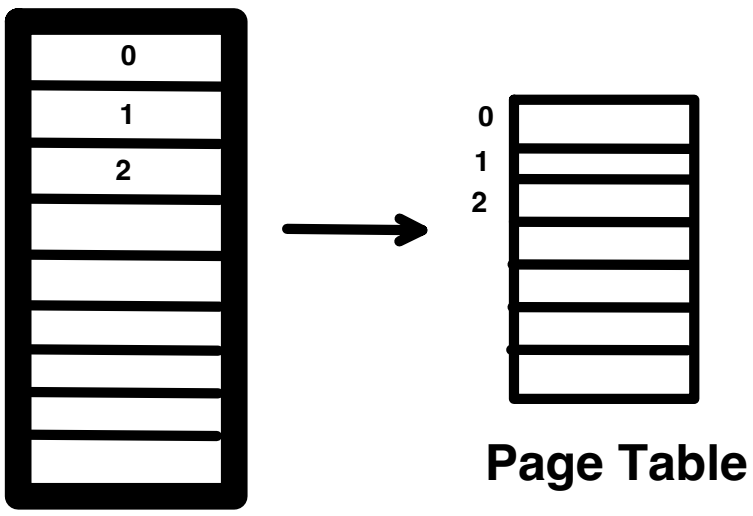
domenica 21 maggio 2023 11:53

56:38

Paginazione a livelli multipli

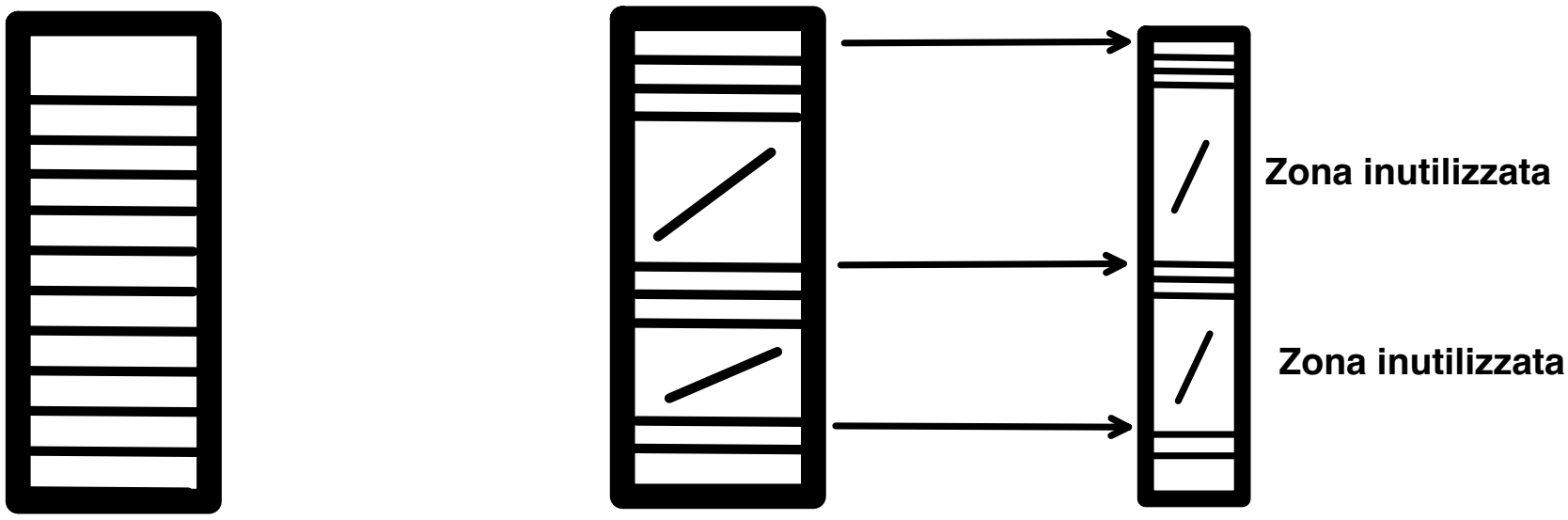
Questo ci fa capire come è strutturata realmente una tabella delle pagine all'interno di un sistema operativo moderno.

Supponiamo di avere un address space e questo address space per sapere se le sue pagine sono realmente materializzate e dove lo sono in RAM, deve essere associato ad una page table. Noi avevamo rappresentato la page table come una tabella in cui avevamo un certo numero di elementi e questo numero di elementi corrispondeva alle eventuali pagine logiche che avevamo all'interno dell'address space. Per la pagina logica zero nell'AB c'era l'elemento zeresimo, per la pagina logica 1 il primo elemento nella page table e così via. Ciascuna delle entry della page table ci andava ad indicare se la pagina era materializzata o non, eventualmente dove era materializzata in RAM.



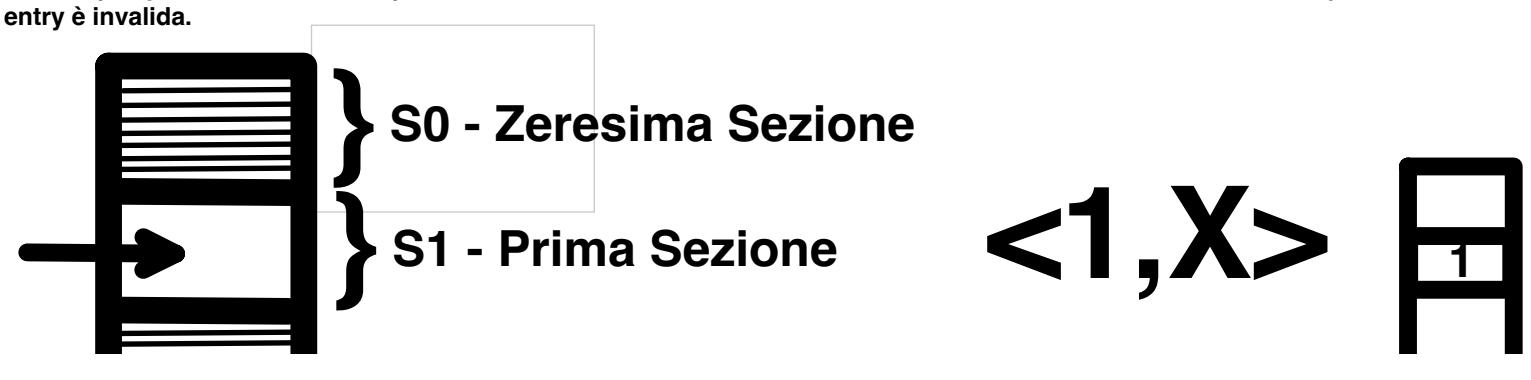
Ma le tabelle delle pagine sono realmente fatte in questo modo?
A livello della CPU, mantenere un indirizzo e, quando la cpu deve accedere a questo indirizzo per andare ad identificare il posizionamento di una pagina logica che eventualmente deve essere acceduta all'interno dell'address space, a partire dall'indirizzo iniziale il sottosistema di controllo non fa altro che partire da quell'indirizzo e spiazzarsi sulla entry.
Funzionano così i processori?
Chiaramente la risposta è no.

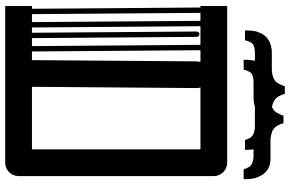
Se noi avessimo una rappresentazione di questo tipo di una page table chiaramente avremmo uno spreco di risorse enormi soprattutto in scenari di questo tipo:
Dato un address space che può anche essere molto grande, e quindi abbiamo una quantità di pagine enormi all'interno. Però supponiamo che questo address space sia di un'applicazione che di pagine ne vuole utilizzare effettivamente poche, e che quindi vi siano mmappate molte poche pagine. Quindi magari REALMENTE sarà fatto con qualche pagina all'inizio, qualche pagina in mezzo e qualche pagina alla fine.



- **Page Table**

Una page table dovrebbe essere strutturata in modo tale da avere una entry per ogni possibile pagina, avremo delle entry per la prima zona, le entry della seconda zona e le entry della terza. Chiaramente però per le pagine che non esistono, la page table non mantiene l'informazione, di conseguenza quella è una zona inutilizzata della page table.
Inutilizzata per andare a determinare il mapping di quelle pagine logiche all'interno della memoria fisica.
Per ottimizzare il tutto sui sistemi operativi si utilizza una paginazione a livelli multipli, che implica dire che quando noi abbiamo l'indirizzo logico in realtà la parte che NON è di offset può essere suddivisa come logicamente in più zone.
Il numero di pagina può essere suddiviso in una parte iniziale P1 e una finale P2.
Cosa vuol dire avere un numero di pagina suddiviso a sua volta in due zone?
La zona P1 ci va ad indicare in quale sezione ci stiamo spostando e la zona P2 ci indica in quale pagina di quale sezione ci stiamo spostando, della sezione identificata tramite P1.
Quindi abbiamo una tabella "di primo livello" che è la directory della tabella delle pagine, che ha una entry per ciascuna possibile sezione specificabile su P1. Se quella sezione è composta da "almeno" alcune pagine per le quali dobbiamo mantenere informazioni di dove queste sono mappate in memoria fisica, eventualmente quella entry punterà ad una seconda tabella, ossia una tabella delle pagine effettive per le pagine della sezione P1.
Ma se noi cerchiamo di accedere ad una pagina all'interno della sezione P1 che non esiste, questo ci porta all'interno della tabella della directory in particolare nella entry associata alla sezione numero 1 e dei bit di controllo andranno ad indicare al processore che tutta quella entry è invalida.





S2 - Seconda Sezione

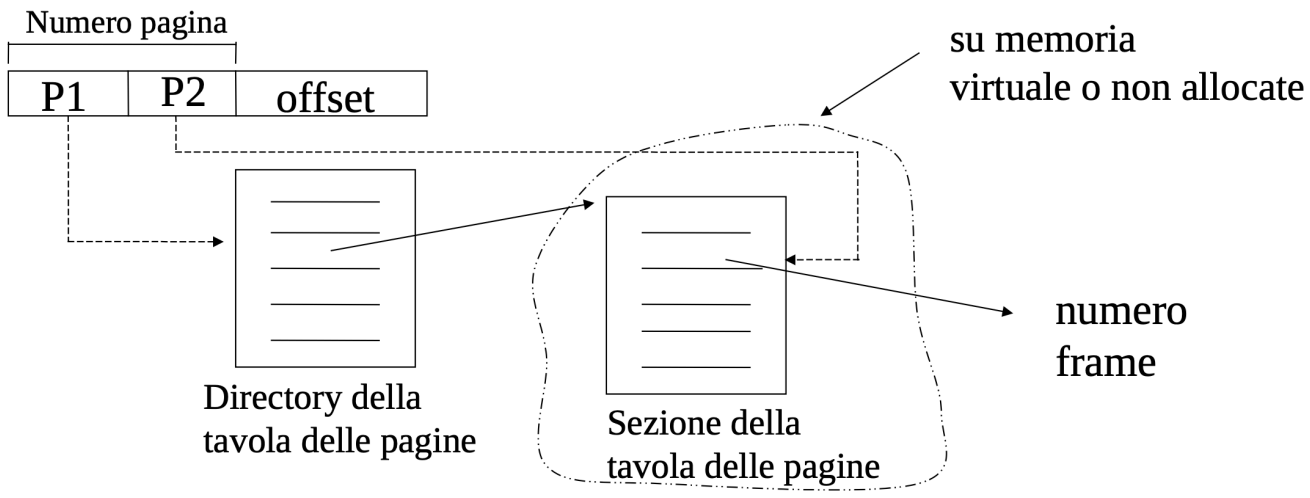
S3 - Terza Sezione

Address Space

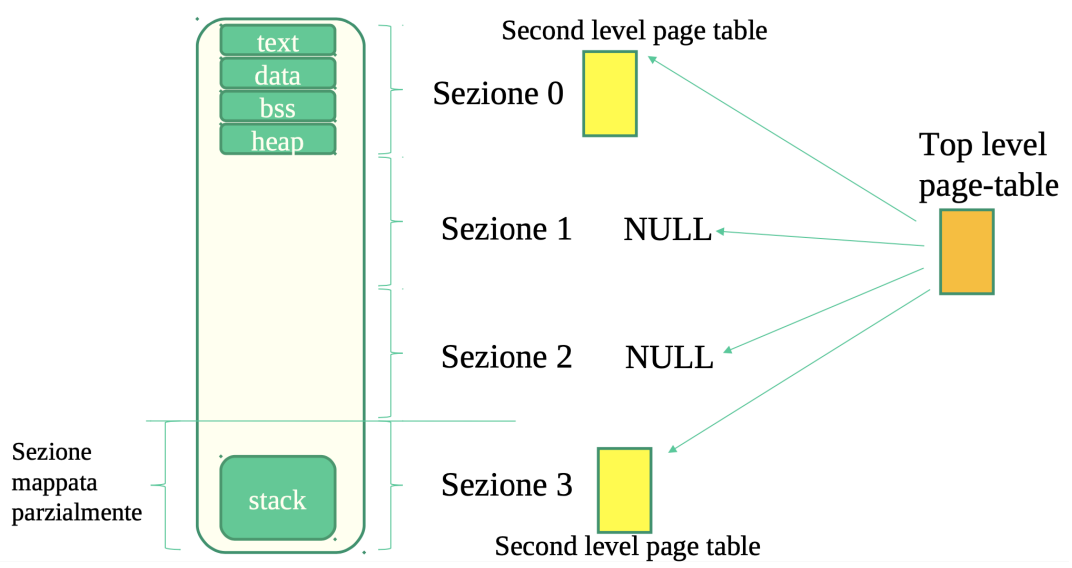
Directory della tabella delle pagine

Ho quattro sezioni e supponiamo di accedere con un pointer ad una pagina che è nella prima sezione, quindi stiamo accedendo alla sezione con indice 1 ad una certa pagina X all'interno di quella sezione.
Quindi andiamo all'interno della tabella della directory nella entry associata alla sezione numero 1 e lì delle informazioni di controllo andranno ad indicare alla CPU che tutta quella entry è invalida. Implica dire che nessuna di quelle pagine è correntemente mappata all'interno della memoria fisica. E quindi, per quella sezione, in quel momento la tabella di secondo livello NON ESISTE.

- le tabelle delle pagine vengono mantenute nello spazio riservato al sistema operativo
- l'occupazione di memoria del sistema operativo può essere ridotta utilizzando la memoria virtuale per la stessa tabella delle pagine o allocando sezioni della tabella solo qualora necessarie



Se noi guardiamo un address space realmente su un sistema operativo moderno, quello che noi abbiamo è che tipicamente è suddiviso in sezioni diverse fatte da un certo numero di pagine:

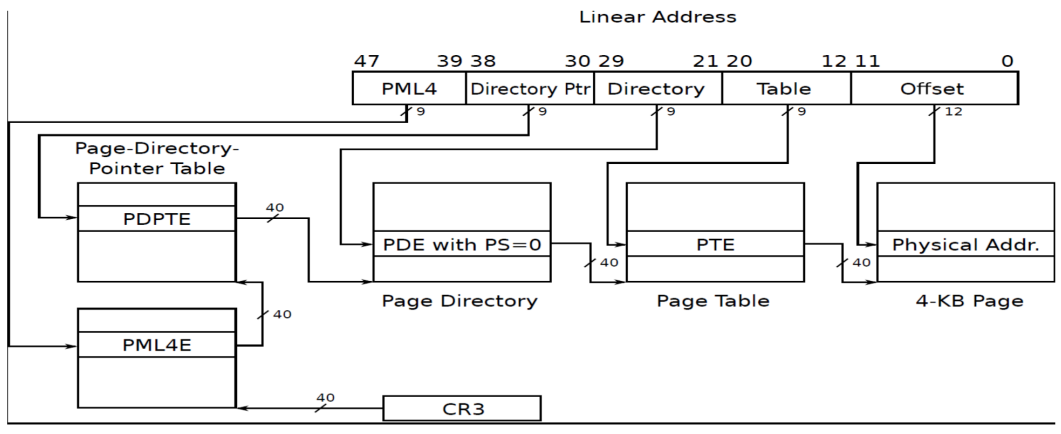


Se in una sezione, come nel caso della sezione 3, abbiamo almeno alcune pagine che sono utilizzabili all'interno di questo address space, per questa sezione una tabella delle pagine ci dovrà essere e dovremmo avere tante entry valide per quante sono queste pagine che possiamo utilizzare all'interno della stack area, ma ovviamente anche tante entry invalide per quante sono le pagine fuori la stack area ma sempre dentro la sezione 3.
Per una sezione in cui non possiamo utilizzare alcuna pagina, il sistema operativo NON alloca in memoria la tabella delle pagine per quella sezione.
Ovviamente all'interno della tabella di primo livello abbiamo semplicemente indichiamo con null se cerchiamo di toccare informazioni che sono presenti all'interno di una sezione che non ha pagine.

Invece di mantenere page table con una entry per ogni possibile pagina che noi potremmo utilizzare all'interno dell'address space, manteniamo le tabelle solo per zone dove eventualmente ci sono delle pagine utilizzabili.

Questo è l'esempio a due livelli, ma se noi andiamo a lavorare su processori moderni i livelli della paginazione sono anche 4.

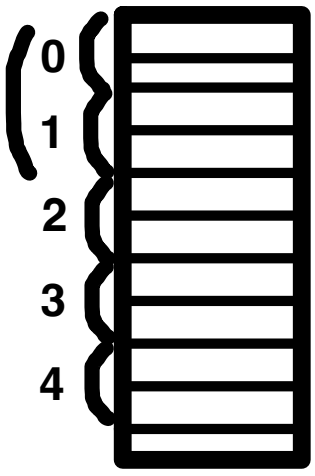
Schema di paginazione su x86-64 (long mode)



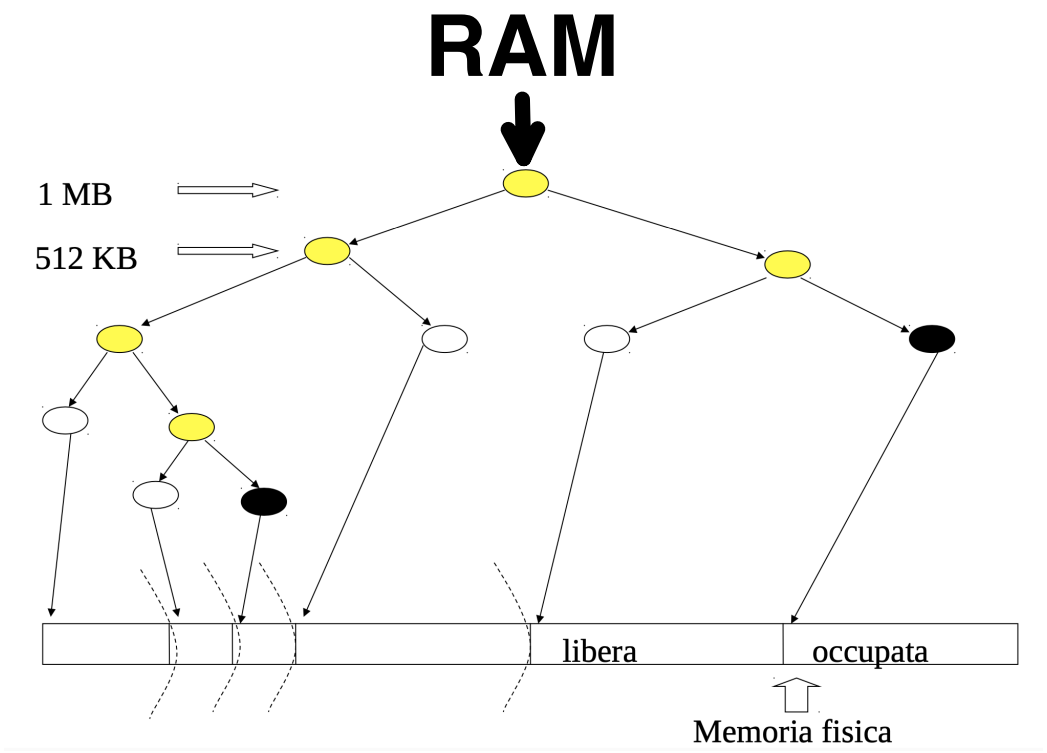
La pagina di primo livello a cui si punta utilizzando il CR3, che è un registro di puntamento della page table corrente, è la PML4.
Da qui se la entry ci dice che si può scendere, per cui ci sono zone possibilmente utilizzabili qui, andiamo in una PDP, e se questa qui ci dice che possiamo scendere andiamo in una PDE e questa, a sua volta, se ci dice che possiamo scendere andiamo all'interno della vera tabella delle pagine per una zona all'interno di questo address space (PTE).
E con la PTE andiamo ad identificare la posizione del FRAME.

Ciascuna di queste tabelle sono composte da 515 elementi ciascuno, tranne l'ultima a destra. 512 alla quarta, per 4kb.
Abbiamo 12 bit per lo spiazzamento e gli altri li abbiamo suddivisi in blocchi di 9 per andare ad indicizzare (di volta in volta) uno degli elementi delle page table che stiamo considerando. Come fa a sapere il kernel del sistema operativo quali sono i frame liberi e quali sono quelli occupati? In generale, il sistema di determinazione di quali sono i frame liberi all'interno della RAM o di quali sono occupati, è il cosiddetto BUDDY SYSTEM.
Sistema dei compagni.
un buddy system funziona in questo modo:

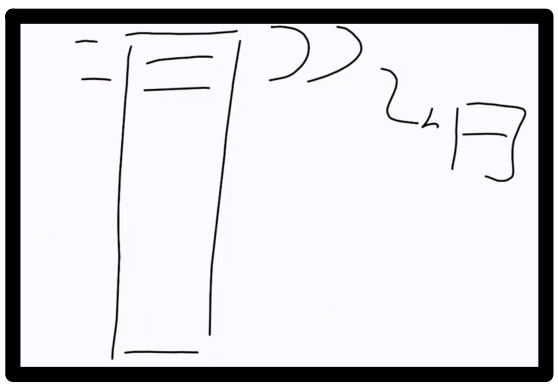
Se noi guardiamo la memoria e la guardiamo attraverso i FRAME, ciascuno di questi frame ha un possibile compagno;
La regola dice che lo zeresimo frame ha il possibile compagno solo nel primo, quindi i primi due possono essere compagni, e quindi a sua volta il primo ha l'unico possibile compagno nel zeresimo.
Quindi potremmo avere che il 2 e il 3 sono possibilmente compagni, 4 e 5 e così via.
Se noi guardiamo questi compagni e li guardiamo in maniera condivisa, il zeresimo e il primo elemento se sono effettivamente compagni rappresentano un'unica area di memoria.
E quand'è che possono rappresentare un'unica area di memoria? Quando sono entrambi liberi, quindi rappresentano un'area di memoria più grande libera composta da entrambi i compagni, oppure quando quell'area di memoria è stata allocata magari per un'informazione che aveva bisogno di due frame.
E se noi globalmente guardiamo le coppie di compagni abbiamo la zeresima coppia e la prima coppia, e questi a sua volta possono essere compagni di ordine di livello superiore.



Questo avverrà quando abbiamo 4 frames completamente liberi, quindi abbiamo una zona di ordine superiore composta da 4 frame, ovviamente allocabile, o dividibile in due zone di ordine inferiore. E ciascuna di queste è divisibile a sua volta in due frame (il zeresimo e il primo) inferiori.
Questo si chiama Buddy System.
A livello di sistema operativo abbiamo una struttura dati che ci tiene traccia dello stato di come è diviso a livello di zone di compagni questo buddy system,



Data la memoria, io mi prendo questi primi due frames, allocando quindi con un ordine che è 2^1, mi prendo questa zona, poi dentro questa zona alloco dei buffer più piccoli che possono essere utilizzati per strutture dati di sistema operativo.



- i blocchi di memoria allocati/deallocati hanno dimensione 2^k dove il minimo è PAGE_SIZE
- max blocco è di taglia 2^u, corrispondente alla memoria totale disponibile per il kernel
- per ogni richiesta di dimensione del blocco pari a s tale che $2^{k-1} < s \leq 2^k$ il blocco di pagine di taglia 2^k viene allocato per la richiesta
- se la precedente condizione non è soddisfatta, allora il blocco di taglia 2^k viene diviso in due e la condizione viene rivalutata sulle taglie 2^{k-1} e 2^{k-2}
- per tenere traccia di blocchi liberi/occupati viene usata una struttura ad albero (o altre strutture dati)
- blocchi adiacenti vengono ricompattati secondo una politica lazy (pigra), ovvero quando il loro numero supera una certa soglia

Dobbiamo determinare quanta porzione all'interno del buddy allocator è necessaria per allocare. Se la quantità di memoria libera è attualmente più del doppio di quella ci serve, la dividiamo a metà e abbiamo due compagni: uno completamente libero e uno e uno che lo andremo ad occupare.