Lezione 9

Gestione delle pagine fisiche

Sistemi operativi

15 maggio 2012

Marco Cesati

System Programming Research Group
Università degli Studi di Roma Tor Vergata

Di cosa parliamo in questa lezione?

Allocazione e rilascio delle pagine di memoria fisiche

1. Frammentazione esterna ed interna della memoria
2. L'allocatore di tipo Buddy system
3. L'allocatore "slab allocator"
4. Pagine fisiche e pagine virtuali
5. Paginazione su richiesta
6. Condivisione delle pagine fisiche tra processi
7. Copiatura su scrittura delle pagine fisiche
8. Gestione delle pagine fisiche in Linux

Allocazione della memoria fisica

Un allocatore di memoria è un componente del SO che si occupa di gestire richieste di blocchi di memoria fisica provenienti da applicazioni utente e programmi del kernel

- In tutti i SO moderni si preferisce ricorrere alla paginazione per gestire la memoria centrale
- Perciò l'unità di base di allocazione della memoria fisica è la page frame
- Alla base del funzionamento di un allocatore di memoria c'è una struttura di dati che memorizza lo stato libero o in uso per ciascuna page frame del sistema
- Gli algoritmi utilizzati negli allocatori sono sofisticati
  - per ridurre la frammentazione della memoria
  - per massimizzare il guadagno di prestazioni dato dalla memoria cache
  - per minimizzare l'overhead di ripetute allocazioni e rilasci

La frammentazione della memoria

Durante la loro esecuzione applicazioni utente e programmi del kernel richiedono e rilasciano continuamente page frame

Gli allocatori di memoria sono progettati principalmente per ridurre un effetto deleterio delle sequenze “casuali” di richieste e rilascio

Frammentazione della memoria

Stato di occupazione della memoria fisica per cui non è possibile assegnare un blocco di memoria fisica richiesto anche se in RAM vi è un numero sufficiente di celle di memoria non utilizzate

In effetti esistono due meccanismi diversi che portano alla frammentazione della memoria:

- La frammentazione esterna
- La frammentazione interna

Frammentazione esterna

Frammentazione esterna della memoria

Stato di occupazione della memoria fisica per cui non è possibile assegnare un blocco di memoria costituito da diverse page frame contigue anche se in RAM vi è un numero sufficiente di page frame non utilizzate

La frammentazione esterna è un fenomeno molto comune che può portare a sprecare molta RAM altrimenti disponibile del sistema

L'*indice di frammentazione esterna* è definito come

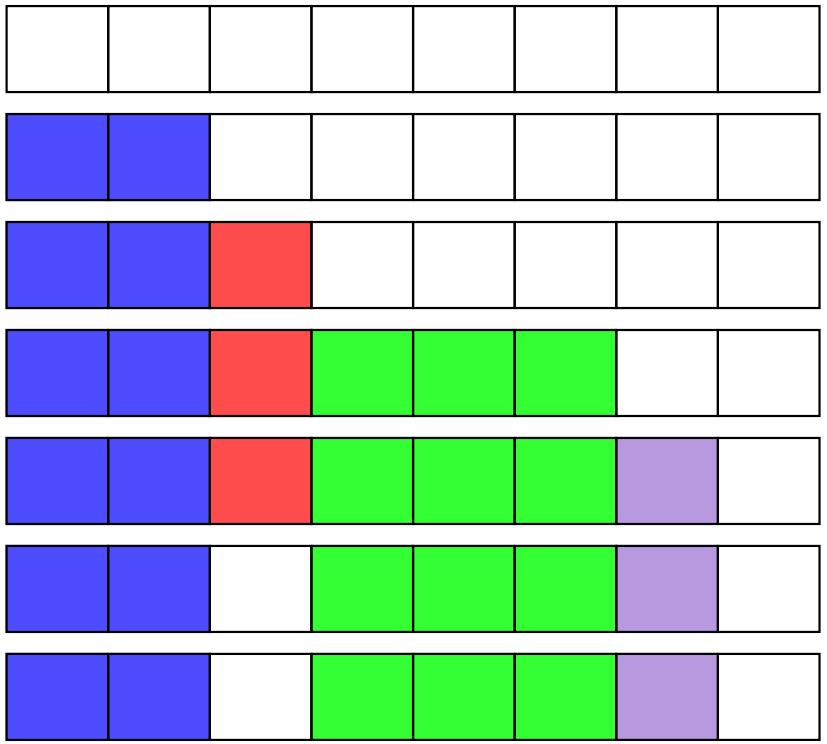
$$1 - \frac{\text{blocco di memoria libera più grande}}{\text{memoria libera totale}}$$

In un sistema vi sono 100 MiB di memoria fisica libera con un indice di frammentazione pari al 95%. Qual è la dimensione del più grande blocco di memoria fisica assegnabile? ***5 MiB***

Esempio di frammentazione esterna

Allocatore di memoria di tipo *first fit*, otto page frame a disposizione

Oper.	Free	Frag.
(inizio)	8 pf	0%
`get(2 pf)`	6 pf	0%
`get(1 pf)`	5 pf	0%
`get(3 pf)`	2 pf	0%
`get(1 pf)`	1 pf	0%
`put(1 pf)`	2 pf	50%
~~`get(2 pf)`~~	2 pf	50%



Frammentazione interna

Frammentazione interna della memoria

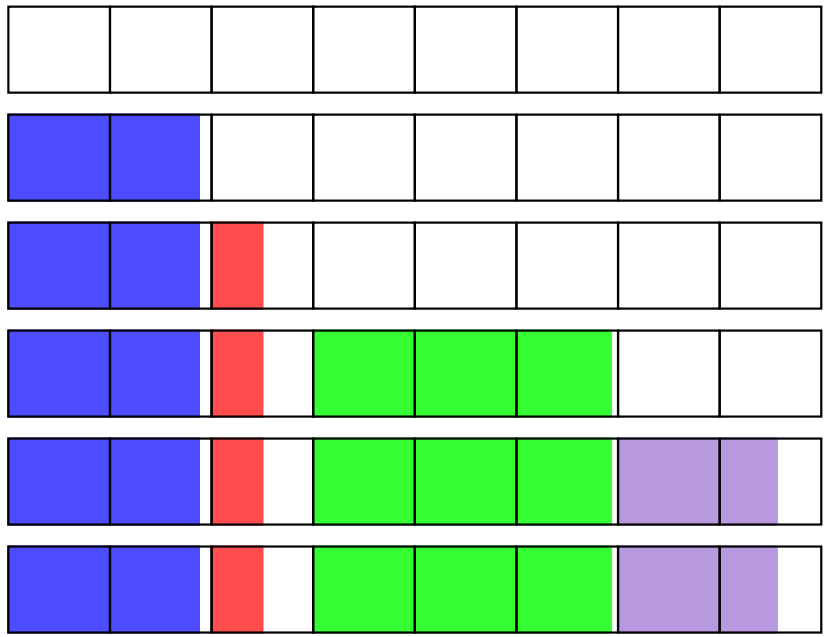
Stato di occupazione della memoria fisica per cui non è possibile assegnare un blocco di memoria anche se in RAM esiste un numero sufficiente di celle non utilizzate poste all'interno di blocchi precedentemente assegnati

- La frammentazione interna è causata principalmente dalle assegnazioni di blocchi di memoria in unità pari alla dimensione della page frame
- Una richiesta per un blocco di dimensione non uguale alla page frame comporta lo "spreco" di celle di memoria finali in una page frame
- Contribuiscono alla frammentazione interna anche le richieste da parte delle applicazioni di blocchi di memoria che non verranno mai effettivamente utilizzati

Esempio di frammentazione interna

Allocatore di memoria di tipo *first fit*, otto page frame a disposizione, page frame di 1024 byte

Oper.	Free
(inizio)	8 pf (8192)
`get(1920)`	6 pf (6272)
`get(512)`	5 pf (5760)
`get(3000)`	2 pf (2760)
`get(1600)`	0 pf (1160)
~~`get(1024)`~~	0 pf (1160)



Gestione della frammentazione

Tutti i SO debbono adottare meccanismi per ridurre gli effetti della frammentazione della memoria

Possibili soluzioni:

- Algoritmi di allocazione progettati per ridurre la frammentazione esterna
  - Buddy system
- Algoritmi di allocazione progettati per ridurre la frammentazione interna
  - *Slab allocator* per le richieste dal kernel
  - *Paginazione su richiesta* per le applicazioni utente
- Procedure di deframmentazione della memoria
  - Tipicamente solo per frammentazione esterna causata dalle applicazioni utente

Buddy system

L'allocatore di memoria Buddy system è stato inventato nel 1963 da H. Markowitz e pubblicato nel 1965 da K.C. Knowlton

- Vengono allocati blocchi di memoria fisica contigua costituiti da un numero intero di page frame
- Lo scopo è ridurre la frammentazione esterna scegliendo opportunamente la posizione dei blocchi assegnati
- È possibile assegnare esclusivamente blocchi memoria costituiti da un numero di page frame uguale ad una potenza di due:

Ordine allocazione:	0	1	2	3	4	5	6	...
Lunghezza in p.f.:	1	2	4	8	16	32	64	...

Qual è il più significativo svantaggio del Buddy system?

L'allocazione di blocchi di grandi dimensioni causa facilmente frammentazione interna

Buddy system (2)

Due blocchi di page frame sono detti *buddy* (gemelli) se:

- I blocchi hanno uguale dimensione b (una potenza di due)
- Le page frame dei due blocchi sono fisicamente contigue in memoria
- Il numero (PFN) della prima pagina del primo blocco è divisibile per $2 \cdot b$

Esempi (un insieme contiene i PFN di un blocco):

$$\{0\} + \{1\} \quad \text{: buddy di ordine 0}$$
$$\{1\} + \{2\} \quad \text{: non buddy (1 non divisibile per 2)}$$
$$\{2,3\} + \{5,6\} \quad \text{: non buddy (non contigui)}$$
$$\{2,3\} + \{4,5\} \quad \text{: non buddy (2 non divisibile per } 2 \cdot 2\text{)}$$
$$\{0,1\} + \{2,3\} \quad \text{: buddy di ordine 1}$$
$$\{4,\ldots,7\} + \{8,\ldots 11\} \quad \text{: non buddy (4 non divisibile per } 2 \cdot 4\text{)}$$
$$\{8,\ldots,11\} + \{12,\ldots 15\} \quad \text{: buddy di ordine 2}$$
$$\{0,\ldots 15\} + \{16,\ldots 31\} \quad \text{: buddy di ordine 4}$$

Allocazione di un blocco nel Buddy system

L'allocatore mantiene per ogni ordine di allocazione una lista contenente blocchi di page frame libere

Poiché le page frame sono libere è possibile usare le page frame stesse per memorizzare i puntatori della lista

Ricerca di un blocco di dimensione b

$d \leftarrow b$
while (la lista per i blocchi di dimensione d è vuota)
    $d \leftarrow 2 \cdot d$
rimuovi il primo blocco B dalla lista ($|B| = d$)
while ($d > b$)
    considera i due blocchi buddy B_1 e B_2 che compongono B
    inserisci B_1 nella lista per la dimensione $d/2$
    $B \leftarrow B_2, \quad d \leftarrow d/2$
assegna il blocco B ($|B| = b$)

Se nel primo ciclo viene superata la massima dimensione ammissibile per un blocco si restituisce "Out of memory"

Rilascio di un blocco nel Buddy system

L'operazione di rilascio di un blocco al Buddy system segue la logica inversa della allocazione, ed opera per quanto possibile la fusione (coalescing) di blocchi

Rilascio di un blocco B di dimensione b

$d \leftarrow b$
while (la lista per la dimensione d contiene il buddy B' di B)
    rimuovi B' dalla lista per la dimensione d
    $B \leftarrow \{B; B'\}$ /* fusione */
    $d \leftarrow 2 \cdot d$
inserisci B nella lista per la dimensione d

È possibile che la lista di dimensione d sia vuota anche se esistono d page frame libere contigue? ***Sì!***

Le d page frame potrebbero non essere divisibili in due blocchi "buddy" di dimensione $d/2$

L'algoritmo è comunque veloce e abbatte la frammentazione esterna

Impiego del Buddy system

- Se una applicazione utente richiede un grande blocco di memoria, il kernel alloca singole page frame (allocazione del Buddy system di ordine 0)
  - Per l'applicazione utente ciò che importa è che il blocco sia contiguo nello spazio di indirizzi virtuali, non in quello fisico
- Quando il kernel del SO necessita di una grande area di memoria fisicamente contigua, vengono effettuate allocazioni del Buddy system di ordine maggiore di 0
- Ciò avviene essenzialmente in due occasioni:
  - Per allocare buffer dedicati al trasferimento di dati DMA
  - Per allocare area di memoria da utilizzare con un altro allocatore di memoria

Perché si utilizzano altri allocatori oltre al Buddy system?

Ad esempio perché allocare aree di memoria fisicamente contigua di grande dimensione (ma non potenza di due) provoca uno spreco di memoria per la frammentazione interna

Lo slab allocator

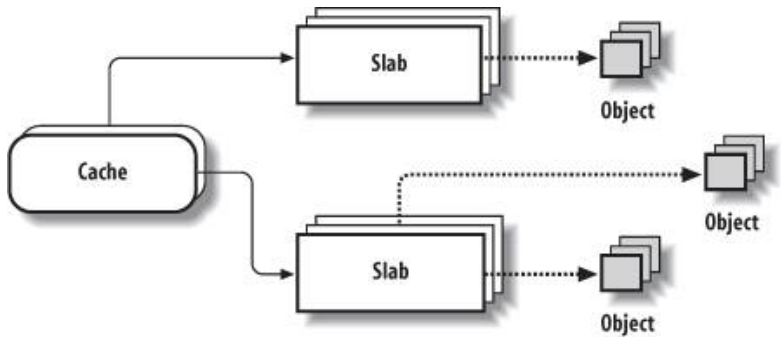
Lo slab allocator è un algoritmo di allocazione della memoria fisica introdotto per la prima volta in Sun Solaris 2.4 (1994)

È basato sulle seguenti premesse:

- Il tipo di dati per il quale una porzione di memoria è allocata può determinare la migliore modalità di allocazione e rilascio
  - Per lo slab allocator le aree di memoria sono *oggetti* consistenti in un insieme di strutture di dati ed un paio di metodi: un *costruttore* ed un *distruttore*
- Le funzioni del kernel tendono a richiedere e rilasciare oggetti dello stesso tipo in continuazione
  - Ad esempio, ogni volta che si crea un nuovo processo è necessario allocare tutti gli oggetti associati al nuovo PCB
  - Lo slab allocator salva gli oggetti rilasciati in una cache: successive richieste per lo stesso tipo di oggetto possono essere evase con estrema rapidità
- Alcuni oggetti (o dimensioni di blocchi di memoria) sono richiesti con altissima frequenza, pertanto è cruciale minimizzare l'impatto della frammentazione interna

Lo slab allocator (2)

- Lo slab allocator è organizzato in *cache*
- Ciascuna cache è un “magazzino” di oggetti dello stesso tipo (in particolare della stessa dimensione)
- Ciascuna cache è costituita da uno o più *slab* (“lastre”)
- Ciascuno slab è un insieme di page frame fisicamente contigue
- In ogni slab sono memorizzati un certo numero di oggetti



Source: D.P. Bovet, M. Cesati, Understanding the Linux kernel, O’Reilly 2005

- Ciascuno slab può essere
  - pieno: tutti gli oggetti sono in uso
  - vuoto: tutti gli oggetti sono liberi
  - parzialmente pieno: solo alcuni oggetti sono in uso

Lo slab allocator (3)

In che modo lo slab allocator riduce la frammentazione interna?

Si può calcolare la dimensione di ciascuno slab in modo da ridurre al minimo la memoria interna sprecata

Ad esempio, se una cache deve contenere oggetti lunghi 592 byte e la pagina è di 4096 byte:

dim. slab	# oggetti	mem. persa
1 pf	6	544 B (13%)
2 pf	13	496 B (6%)
3 pf	20	448 B (3.6%)
4 pf	27	400 B (2.4%)

È possibile sfruttare parte dei byte “sprecati” entro lo slab per:

- memorizzare strutture di dati necessarie allo slab allocator (ad esempio, lo stato libero/occupato degli oggetti)
- disallineare gli indirizzi iniziali degli oggetti per sfruttare meglio la memoria cache hardware (*slab coloring*)

Allocazione e rilascio con lo slab allocator

Allocazione di un oggetto

Se la cache contiene uno slab parzialmente pieno o vuoto
- Marca un oggetto libero dello slab come occupato
- Aggiorna se necessario lo stato dello slab
- Termina restituendo l'indirizzo fisico dell'oggetto

Se tutti gli slab della cache sono pieni
- Alloca un nuovo slab con una richiesta al Buddy system
- Inizializza gli oggetti nello slab
- Ripeti l'intera procedura di allocazione

Rilascio di un oggetto

Marca l'oggetto nella cache come libero
Aggiorna se necessario lo stato dello slab

La memoria occupata dagli oggetti liberi dello slab allocator non è in realtà libera per il Buddy system

In caso di necessità un algoritmo di recupero della memoria ricerca gli slab vuoti e rilascia le page frame al Buddy system

Utilizzo degli indirizzi virtuali

I sistemi operativi moderni sfruttano in modo essenziale i meccanismi hardware di traduzione da indirizzi virtuali a fisici

- I processi utilizzano uno spazio di indirizzamento virtuale indipendente dalle caratteristiche hardware della RAM
  - Maggiore semplicità e portabilità dei programmi
  - Possibilità di eseguire un processo che richiede più memoria della quantità di memoria fisica a disposizione
- Il sistema operativo gestisce in modo indipendente la memoria fisica e la memoria assegnata ai processi
  - Maggiore semplicità ed efficienza del sistema
  - Possibilità di attivare ed eseguire molti processi le cui richieste di memoria globalmente superano la quantità di memoria fisica installata nel sistema
- Il SO può stabilire associazioni tra il contenuto della memoria virtuale e quello di file in memoria secondaria
  - Maggiore efficienza nel trasferimento dei dati da disco
  - Possibilità di salvare su disco pagine di memoria virtuale

Informazioni associate ad una pagina virtuale

In una architettura con tabelle di paginazione dirette l'indirizzo virtuale iniziale di una pagina è associato ad una voce in una tabella di paginazione tipicamente contenente:

- Un flag che indica se la pagina virtuale è associata ad una page frame (*bit di validità*)
- Eventualmente, il numero di page frame (PFN) associata
- I diritti di accesso (lettura, scrittura, esecuzione)
- Eventualmente, la posizione della pagina virtuale in memoria secondaria
- Altri flag utili al sistema operativo

Come si identifica il processo "proprietario" di una pagina?

- Se la pagina è intesa come gruppo di indirizzi virtuali, non esiste alcun processo proprietario
- Ciascun processo mappa la pagina virtuale su una pagina fisica tramite una voce nelle proprie tabelle di paginazione

Eccezione di pagina mancante

Un evento molto comune durante l'esecuzione di un programma in un SO con memoria virtuale è il page fault

Eccezione di pagina mancante (page fault trap)

Eccezione hardware che interrompe il flusso di esecuzione di un programma quando una istruzione macchina tenta di accedere ad una cella di memoria il cui indirizzo fisico non può essere determinato dalla MMU

Tipicamente il page fault è generato dalla MMU quando deve tradurre un indirizzo virtuale ed una delle voci delle tabelle di paginazione coinvolte ha il bit di validità non impostato

Nei SO con memoria virtuale il page fault non corrisponde necessariamente al malfunzionamento di software o hardware

Se il SO pone rimedio alla causa del page fault, l'istruzione macchina è rieseguita ed il processo continua l'esecuzione

Paginazione su richiesta

Si consideri il caricamento in memoria di un file eseguibile residente su memoria secondaria

Il SO potrebbe assegnare immediatamente al processo tutte le page frame occorrenti per l'esecuzione ed inizializzarle con il codice macchina ed i dati trasferiti dal disco

Quali sono gli svantaggi di questo approccio?

- Il processo potrebbe eseguire solo una piccola parte del codice macchina definito nel file eseguibile
- Il processo potrebbe accedere solo ad una piccola parte dei dati
- Il tempo necessario per trasferire tutte le pagine dalla memoria secondaria può essere significativamente lungo

I SO moderni adottano una tecnica detta paginazione su richiesta (demand paging) che consiste nel ritardare il più possibile l'effettiva allocazione di page frame al processo

Paginazione su richiesta (2)

Lo schema di funzionamento della *paginazione su richiesta* (*demand paging*):

- Una applicazione invoca una chiamata di sistema per richiedere alcune pagine di memoria
- Il kernel annota nel PCB che lo spazio di indirizzamento virtuale del processo include le nuove pagine
- La chiamata di sistema restituisce al processo l'indirizzo virtuale iniziale delle nuove pagine
- L'esecuzione del processo continua normalmente fino a quando esso tenta di accedere ad una delle nuove pagine
- Poiché le tabelle di paginazione del processo non sono ancora state modificate, la MMU genera un page fault
- Il kernel alloca una singola page frame e registra il suo indirizzo nella voce della tabella di paginazione
  - Se necessario il kernel inizializza il contenuto della nuova pagina, ad esempio trasferendolo da memoria secondaria
- Il processo continua l'esecuzione rieseguendo l'istruzione macchina che accede alla nuova pagina

Vantaggi della paginazione su richiesta

- La paginazione su richiesta è un meccanismo completamente trasparente per l'applicazione utente
  - L'istruzione macchina che genera il page fault trap è rieseguita automaticamente al termine dell'assegnazione della page frame
- Il kernel assegna ai processi esclusivamente le page frame accedute effettivamente, e quindi effettivamente necessarie per l'esecuzione
  - Spesso le applicazioni richiedono blocchi di memoria che non verranno in tutto od in parte utilizzati
- Il kernel evita i trasferimenti dalla memoria secondaria corrispondenti alle pagine che l'applicazione non utilizzerà

Condivisione delle page frame

- La possibilità di gestire singolarmente ogni singola page frame di un processo permette di realizzare facilmente meccanismi di condivisione delle page frame
- Programmi differenti utilizzano spesso gli stessi dati costanti e lo stesso codice macchina
  - Esempio: libreria di sistema
- Se una applicazione richiede memoria che non deve essere modificata, il kernel assegnerà page frame marcate nelle tabelle di paginazione come di sola lettura

Condivisione delle page frame

Meccanismo che consente la condivisione di una medesima page frame senza diritti di scrittura (non modificabile) da parte di più processi differenti

In pratica lo stesso numero di page frame (PFN) appare più volte nelle tabelle di paginazione di differenti processi

Il kernel tiene traccia con appositi contatori del numero di processi che condividono ciascuna page frame

Copiatura su scrittura

- Nei sistemi operativi di tipo Unix il processo di creazione di un nuovo processo prevede la duplicazione dell'intera memoria del processo padre
- Poiché il numero di page frame di un processo può essere elevato, questa operazione è potenzialmente costosa
- Oltre tutto l'operazione di duplicazione è inutile nel caso in cui alla `fork()` segua immediatamente una `execve()`, che prevede il rilascio di tutte le page frame del processo
- Se una pagina ha soltanto diritti di lettura e/o esecuzione, allora la page frame può essere inserita in condivisione nelle tabelle di paginazione del padre e del figlio
- Ma cosa fare con la pagine con diritti di scrittura?

Copiatura su scrittura (copy on write, COW)

Meccanismo che consente la condivisione delle page frame con diritti di scrittura tra diversi processi basato sulla manipolazione delle voci delle tabelle di paginazione

Copiatura su scrittura (2)

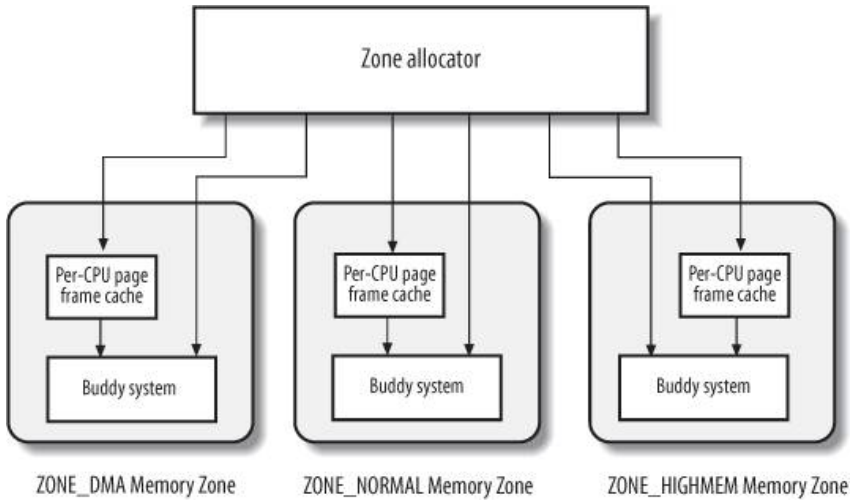
Schema del copy on write in seguito ad una `fork()`:

- Tutte le page frame del processo padre vengono inserite nelle tabelle di paginazione del processo figlio
  - In effetti vengono duplicate le tabelle di paginazione
- A tutte le voci delle tabelle di paginazione sia del padre che del figlio vengono tolti i diritti di scrittura, e vengono incrementati i contatori di condivisione
- Ogni accesso in scrittura ad una page frame COW provoca un page fault e l'intervento del kernel
- Il kernel controlla il contatore di condivisione della pagina e se necessario:
  - richiede al Buddy system una nuova page frame
  - effettua la copia della page frame
  - attribuisce diritti di scrittura alla nuova pagina
  - sostituisce il nuovo PFN nella tabella di paginazione

Esempio: gestione delle pagine fisiche in Linux

Il kernel Linux ha una complessa gestione delle pagine fisiche:

- Le pagine di memoria fisica sono suddivise in *nodi* (per il supporto ai sistemi NUMA)
- In ciascun nodo sono definite diverse *zone*
  - Page frame in zone differenti debbono essere trattate in modo diverso dal kernel
- In ciascuna zona e per ciascuna CPU si trova una piccola cache di page frame libere e pronte all'uso
- In ciascuna zona è definito un allocatore di memoria di tipo Buddy system



Source: D.P. Bovet, M. Cesati, Understanding the Linux kernel, O'Reilly 2005

Esempio: gestione delle pagine fisiche in Linux (2)

- Al di sopra dei Buddy system è utilizzato un allocatore di tipo slab allocator
- In effetti esistono tre diverse implementazioni dello slab allocator integrate nel codice standard:
  - SLAB: modellato sull'allocatore tradizionale
  - SLUB: più veloce e scalabile (oggi default)
  - SLOB: ottimizzato per sistemi con poca memoria
- Al di sopra dello slab allocator è utilizzato *kmalloc*: un allocatore di memoria fisicamente contigua
- Al di sopra di kmalloc è utilizzato *vmalloc*: un allocatore di memoria virtualmente contigua

Nel kernel Linux sono anche utilizzate procedure per recuperare pagine di memoria per nuove allocazioni, migrare pagine di memoria fisica tra i nodi, deframmentare la memoria, cercare page frame identiche da gestire con COW, ...