Sistemi Operativi

Ionut Zbirciog

14 November 2023

ALGORITMI DI SOSTITUZIONE DELLE PAGINE

Quando si verifica un page fault, per far spazio alla pagina entrante il sistema operativo deve scegliere una pagina da sfrattare (rimuovere dalla memoria). Se la pagina da rimuovere è stata modificata mentre era in memoria, deve essere riscritta sulla memoria non volatile per aggiornare la copia su disco o SSD. Se invece la pagina non è stata modificata, la copia su disco o SSD è già aggiornata e non c'è bisogno di riscrivere. La pagina da leggere sovrascrive semplicemente la pagina sfrattata.

Sarebbe possibile prendere una pagina a caso da rimuovere a ogni page fault, ma le prestazioni del sistema migliorano molto se si sceglie una pagina non particolarmente utilizzata. Se si sfrattasse una pagina usata frequentemente, con ogni probabilità dovrebbe essere riportata in memoria a breve, con il risultato di un ulteriore sovraccarico.

1. L'algoritmo ottimale di sostituzione delle pagine

Ciascuna pagina può essere etichettata con il numero di istruzioni da eseguire prima di ricevere un riferimento per la prima volta. L'algoritmo di sostituzione ottimale indica che deve essere rimossa la pagina con l'etichetta con il numero più alto. Se una pagina non sarà usata per 8 milioni di istruzioni e un'altra pagina per 6 milioni di istruzioni, rimuovere la prima allontana il page fault che la ricaricherebbe il più in là possibile.

Il solo problema di questo algoritmo è che è irrealizzabile. Al momento del page fault, il sistema operativo non ha modo di sapere in quale momento verrà fatto il prossimo riferimento a ciascuna delle pagine. Tuttavia, eseguendo un programma su un simulatore e tenendo traccia di tutti i riferimenti alle pagine, è possibile implementare una sostituzione delle pagine ottimale sulla seconda esecuzione, usando le informazioni sui riferimenti alle pagine raccolte durante la prima esecuzione. In questo modo è possibile confrontare le prestazioni degli algoritmi realizzabili con quelle del migliore algoritmo possibile.

2. Not Recently Used (NRU)

Al fine di consentire al sistema operativo la raccolta di statistiche utili sull'uso delle pagine, molti computer con memoria virtuale hanno due bit di stato, R e M, associati a ciascuna pagina. R viene impostato quando si fa riferimento alla pagina (in lettura o scrittura). M è impostato quando la pagina viene scritta (cioè modificata). I bit sono contenuti in ciascuna voce della tabella delle pagine. I bit vengono aggiornati dall'hardware ad ogni accesso.

I bit R e M possono essere usati per costruire un algoritmo di paginazione semplice come il seguente. All'avvio di un processo, entrambi i bit di pagina di tutte le pagine sono impostati a 0 dal sistema operativo. Periodicamente (per esempio a ogni interrupt del clock), il bit R è ripulito, per contraddistinguere le pagine che non hanno avuto riferimenti recentemente da quelle che ne hanno avuti. Quando avviene un page fault, il sistema operativo ispeziona tutte le pagine e le divide in 4 categorie basate sui valori attuali dei loro bit R e M:

- Classe 0: nessun riferimento, non modificata.
- Classe 1: nessun riferimento, modificata.
- Classe 2: riferimento, non modificata.
- Classe 3: riferimento, modificata.

Le pagine di classe 1 sembrano a prima vista impossibili, ma appaiono quando un interrupt del clock azzera il bit R di una pagina di classe 3. Gli interrupt del clock non azzerano il bit M perché questa informazione è necessaria per sapere se la pagina deve essere riscritta su disco o meno. Azzerare R ma non M produce una pagina di classe 1. In altre parole, una pagina di classe 1 è stata modificata molto tempo fa e da allora non è stata più toccata.

L'algoritmo NRU (Not Recently Used) rimuove una pagina a caso dalla classe non vuota con il numero più basso. I vantaggi sono: Semplicità, efficienza implementativa e prestazioni accettabili.

3. First-In, First-Out (FIFO)

Descrizione FIFO è un algoritmo di paginazione che elimina la pagina più vecchia in memoria.
Implementazione Il sistema operativo rimuove la pagina in testa alla lista (la più vecchia) durante un page fault, aggiungendo la nuova pagina in coda.
Problema Nel contesto informatico, la pagina più vecchia potrebbe ancora essere frequentemente utilizzata (ricorsione), rendendo FIFO poco efficace.
Conclusione A causa di queste limitazioni, FIFO è raramente utilizzato nella sua forma più semplice.

4. Seconda Chance

Una semplice modifica a FIFO che evita il problema di gettare una pagina usata di frequente consiste nel controllare il bit R della pagina più vecchia. Se è 0, la pagina è vecchia e inutilizzata e viene così sostituita immediatamente. Se R è 1, il bit viene azzerato, la pagina è posta in fondo all'elenco e il momento in cui è stata caricata in memoria viene aggiornato per farla sembrare appena arrivata. Poi la ricerca continua. L'operatività di questo algoritmo, detto seconda chance, è illustrata nella figura.

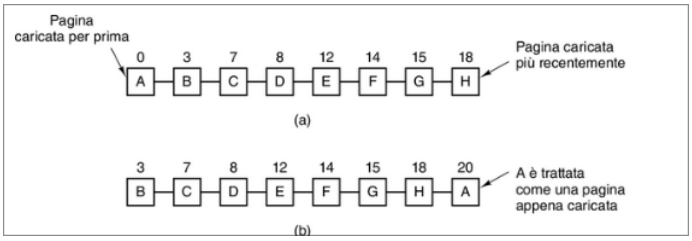

Figure 1: Seconda Chance.

Supponiamo che si verifichi un page fault all'istante 20.

La pagina più vecchia è A, arrivata al momento 0, all'inizio del processo. Se A ha il bit R azzerato, allora è rimossa dalla memoria, sia scrivendola su memoria non volatile (se è sporca) sia semplicemente scaricandola (se è pulita). Se invece il bit R è impostato, A viene portata in fondo alla lista e il suo "momento di caricamento" è reimpostato all'attuale (20). Anche il bit R è azzerato. La ricerca della pagina adatta prosegue con B.

Quello che la seconda chance sta cercando è una vecchia pagina che non sia stata oggetto di riferimenti durante l'ultimo intervallo del clock. Se tutte le pagine hanno avuto riferimenti, la seconda chance degenera in un FIFO puro.

5. Clock

Funzionamento

Lista circolare dei frame di pagina con un puntatore simile ad una lancetta di un orologio per identificare la pagina più vecchia. Quando avviene un page fault, la pagina indicata dalla lancetta viene controllata. Se il suo bit R è 0 viene sfrattata, la nuova pagina viene inserita al suo posto nell'orologio e la lancetta viene spostata in avanti di una posizione. Se R è 1, viene azzerato e la lancetta passa alla pagina successiva. Questo processo è ripetuto finché viene trovata una pagina con R = 0. È più efficiente rispetto a Seconda chance e FIFO.

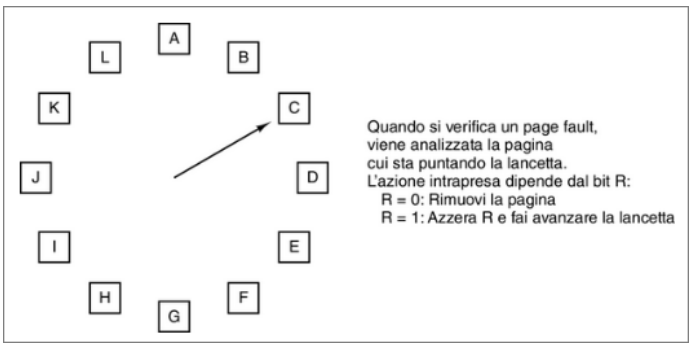

Figure 2: Clock.

6. Least Recently Used (LRU)

L'idea del algoritmo è di sfrattare la pagina rimasta inutilizzata per più tempo. Teoricamente l'algoritmo è fattibile, il problema è l'implementazione. Per implementare tale algoritmo è necessario tenere in memoria una lista concatenata di tutte le pagine, con quelle più usate in testa e quelle meno usate in coda. La difficoltà sta nel fatto che l'elenco deve essere aggiornato a ogni riferimento alla memoria. Trovare una pagina in memoria, cancellarla e poi portarla in testa è un'operazione che costa del tempo, anche se eseguita nell'hardware.

Esistono però altri metodi per implementare l'LRU con hardware speciale. Usare un contatore a 64 bit per ogni riferimento a memoria. Alla generazione di un page fault, si rimuove la pagina con contatore più basso, indicando l'uso meno recente.

7. Simulazione del LRU via software - algoritmo NFU (Not Frequently Used)

Associa ad ogni pagina un contatore, incrementato con ogni interrupt del clock in base al bit R. I contatori tengono traccia del numero di riferimenti di ciascuna pagina. Quando avviene un page fault, la scelta di quale pagina sostituire cade su quella con il contatore più basso.

Il problema principale dell'NFU è che non dimentica mai nulla. Per esempio, nel caso di un compilatore multipass (a molteplici passaggi), le pagine usate frequentemente durante il passaggio 1 avrebbero un conteggio alto anche nei passaggi successivi. Infatti, se accade che il passaggio 1 è quello con il tempo di esecuzione più alto fra tutti i passaggi, le pagine contenenti il codice per i passaggi seguenti potrebbero avere sempre conteggi inferiori alle pagine del passaggio 1, quindi il sistema operativo rimuoverebbe pagine utili invece di pagine non più utilizzate.

Il NFU si può migliorare semplicemente con il concetto di aging. Si usa un numero limitato di bit (8), ad ogni interrupt del clock i bit vengono spostati a destra, ma prima dello shifting, il bit R viene aggiunto a sinistra. Quando avviene un page fault, viene rimossa la pagina con il contatore più basso. È evidente che una pagina che non abbia riferimenti, diciamo, per quattro cicli del clock, avrà 4 zero consecutivi nel suo contatore e avrà così un valore inferiore rispetto a una senza riferimenti per tre cicli del clock.

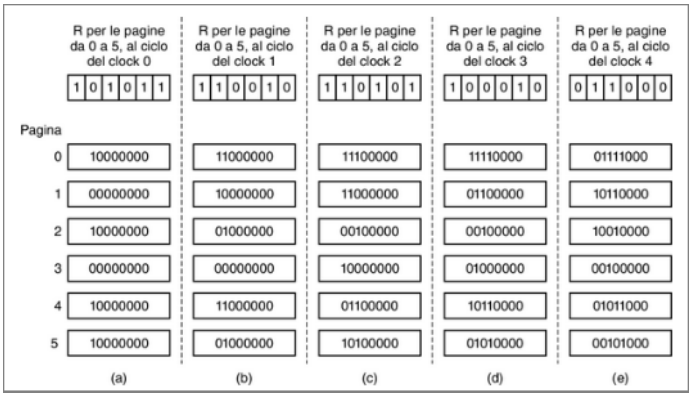

Figure 3: NFU con aging.

Consideriamo come esempio la pagina 1.

a. Non è modificata ed ha valore 0 0 0 0 0 0 0 0.

b. Viene modificata ed ha valore 1 0 0 0 0 0 0 0.

c. Viene modificata ed ha valore 1 1 0 0 0 0 0 0.

d. Non viene modificata ed ha valore 0 1 1 0 0 0 0 0.

Adesso consideriamo le pagine 3 e 5.

c. Entrambe hanno avuto accesso:

3. $3 \rightarrow 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0$
5. $5 \rightarrow 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0$

d ed e. Nessuna delle due ha avuto riferimenti:

3. $3 \rightarrow 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0$
5. $5 \rightarrow 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0$

Con NFU è Aging, la pagina 3 viene rimossa poiché la pagina 5 ha avuto riferimenti in (a) prima e la pagina 3 no. L'uso di un contatore a 16, 32 o 64 bit fornisce una cronologia più lunga, al costo di una maggior quantità di memoria per contenerla. Di solito 8 bit sono più che sufficienti.

8. Working Set

Demand Paging

Quando i processi sono avviati, nessuna delle loro pagine è in memoria. Appena la CPU prova a prelevare la prima istruzione, genera un page fault e fa sì che il sistema operativo prelevi la pagina con la prima istruzione. Rapidamente seguono altri page faults per le variabili globali e lo stack. Dopo un certo tempo, il processo ha la maggior parte delle pagine che gli servono ed è in condizione di essere eseguito con relativamente pochi page faults.

Working Set

L'insieme delle pagine che un processo sta attualmente usando. Se l'intero set di lavoro è in memoria, il processo sarà eseguito senza causare molti page faults fino a quando passerà a un'altra fase dell'esecuzione.

Molti sistemi di paginazione cercano quindi di tenere traccia del set di lavoro di ciascun processo e di accertarsi che sia in memoria prima di consentirne l'esecuzione. Tale approccio è chiamato *working set model* ed è stato progettato per ridurre notevolmente la frequenza dei page faults. Il caricamento delle pagine prima di lasciar eseguire i processi è detto anche prepaginazione o prepaging.

È risaputo che raramente i programmi fanno riferimenti a tutto il loro spazio degli indirizzi uniformemente; i riferimenti tendono a raggrupparsi su un numero ristretto di pagine. In ogni istante t esiste un insieme che consiste di tutte le pagine usate dai k riferimenti alla memoria più

recenti. Questo insieme $w(k,t)$ è il set di lavoro. $w(k,t)$ è una funzione monotona non decrescente di k al crescere di k perché le pagine o aumentano oppure il numero rimane costante se non servono al processo. Il limite di $w(k,t)$ quando k diventa grande è finito, poiché un programma non può fare riferimento a più pagine di quante ne contenga il suo spazio degli indirizzi e pochi programmi utilizzeranno ogni singola pagina.

Se il working set di un processo è completamente in memoria, si verificano pochi page faults. Se il working set è più grande della memoria disponibile, si verificano frequenti page faults, rallentando significativamente il processo. Questo fenomeno è noto come *thrashing*, ovvero la CPU è occupata solo a scambiare le pagine da memoria volatile a memoria non volatile.

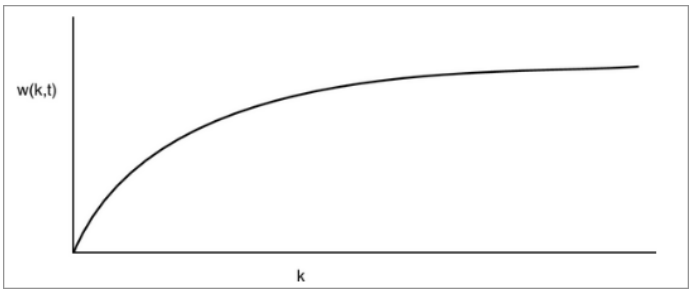

Figure 4: Tracciamento del Working Set.

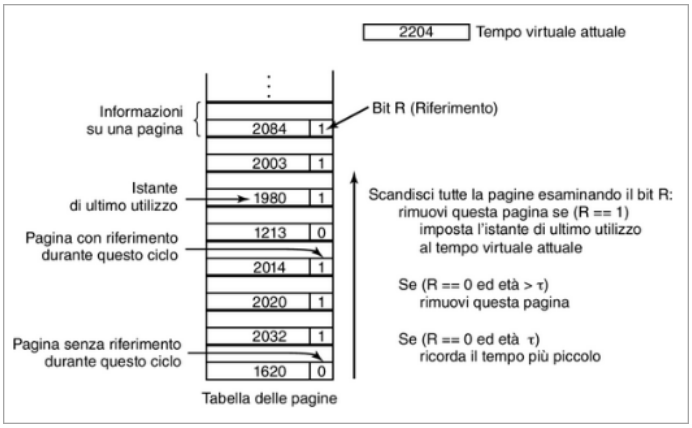

Figure 5: Algoritmo basato su working set.

Andiamo adesso ad analizzare un algoritmo di sostituzione delle pagine che si basa sul set di lavoro. L'idea base è quella di trovare una pagina che non sia nel set di lavoro e rimuoverla. Un interrupt periodico azzera il bit R a ogni ciclo di clock.

Cosa succede durante un page fault? Viene fatta una scansione delle pagine alla ricerca di una pagina da rimuovere. Viene controllato il bit R di ciascuna pagina.

- Se $R = 1$, viene aggiornato il tempo dell'ultimo utilizzo, la pagina rimane nel working set.
- Se $R = 0$ e eta $> t$, la pagina non è nel working set e viene rimossa.

- Se $R = 0$ e eta $\leq t$, la pagina è nel working set e non viene rimossa.

Se nessuna pagina è rimovibile, viene selezionata la più vecchia con $R = 0$, altrimenti una a caso.

9. WSClock

WSClock è un miglioramento dell'algoritmo di Clock che integra le informazioni del working set. Usa una lista circolare di frame, simile all'algoritmo Clock. Ogni frame nella lista contiene: il tempo di utilizzo, il bit R e il bit M.

Ad ogni page fault è esaminata per prima la pagina indicata dalla lancetta dell'orologio. Se il bit R = 1 la pagina non è la candidata ideale alla rimozione, ovvero è stata usata nel ciclo del clock, poi il bit R viene impostato a 0. La lancetta avanza alla pagina successiva e l'algoritmo viene ripetuto per la nuova pagina.

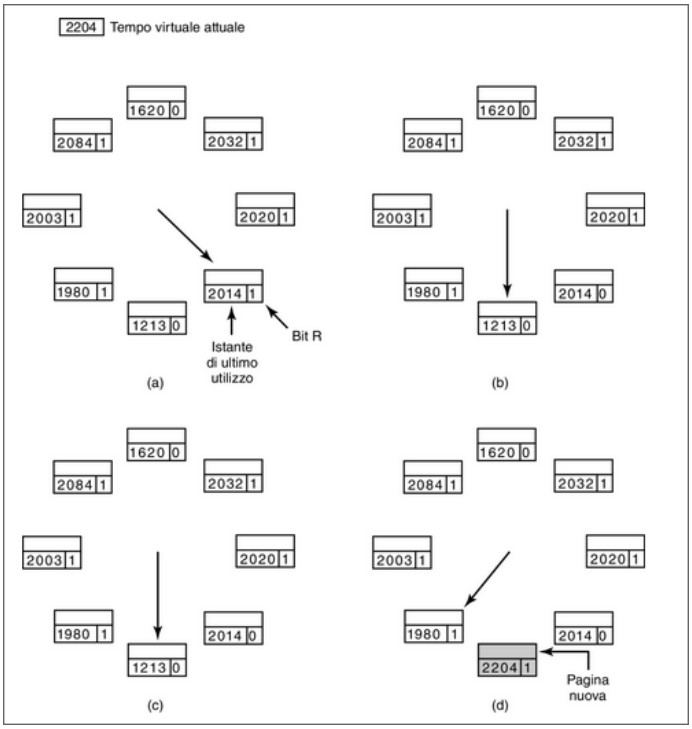

Figure 6: WSClock.

Se la pagina indicata ha $R = 0$ e se eta $> t$:

- Se $M = 0$ (pagina pulita), non è nel working set e ne esiste una copia valida su memoria non volatile. Quindi il frame viene semplicemente riciclato e vi viene posta la nuova pagina.
- Se $M = 1$ (pagina sporca), non ne esiste una copia valida in memoria non volatile quindi non può essere sfrattata immediatamente.

Per evitare "rallentamenti", la scrittura su memoria non volatile viene schedulata e rimandata. Lungo la lista potrebbe esserci una pagina pulita e vecchia che può essere usata immediatamente. La lancetta avanza e l'algoritmo procede con la pagina successiva.

In conclusione, quali sono i migliori algoritmi? Aging e WSClock sono i migliori tra gli algoritmi analizzati, entrambi basati rispettivamente su LRU e sull'idea di Working Set, con buone prestazioni e implementazione efficiente. La nozione di migliore è il risultato del trade-off tra la complessità del metodo e i vincoli hardware che il sistema operativo deve comunque rispettare.