SIS 2024-12-04

SIS

2024-12-04

Contents

1 virtualizzazione **1**

2 Algoritmi di sostituzione delle pagine - pagina vittima **2**
2.1 Algoritmo FIFO 2
2.2 Algoritmo ottimale 2
2.3 Algoritmo Least Recently Used LRU 2
2.4 algoritmi di stack 3
2.5 implementazione LRU 3
2.6 Algoritmo Not Recently Used NRU 3
2.7 Algoritmo Not Frequently Used NFU 3
2.8 Algoritmo aging 4
2.9 Algoritmo a orologio CLOCK - second chance 4
2.10 CLOCK migliorato 4

3 Trashing **4**
3.1 modello di working-set 5
3.2 algoritmi di allocazione 5

1 virtualizzazione

cruciali algoritmi di scelta pagina vittima per minimizzare il numero di page fault, il tempo di accesso alla memoria secondaria è di ordine di grandezza maggiore rispetto a quello della memoria principale.
La memoria virtuale porta a diversi benefici, permette di superare lo spazio di memoria fisico, permette di incrementare la multiprogrammazione e favorisce la nascita dei processi (es copy-on-write).
fork, copia spazio indirizzi del padre, nella virtual fork invece si permette condivisione spazio indirizzi, mentre programmatore evita inconsistenze. Invece il copy-on-write permette di condividere la stesse pagine di memoria, ma una volta acceduta in scrittura la pagina viene duplicata. Più rapido perché alla creazione condividi lo spazio indirizzo ma non lo duplichi.

UN'altra tecnica utilizzata è quella relativa al memory-mapped I/O, che permette di mappare un file in memoria secondaria, (...), quando blocco file viene acceduto prima volta il blocco viene portato in memoria principale, in modo da velocizzare l'accesso, semplificando gestione dell'input output. Permette anche gestione condivisione file tra più processi mediante condivisione delle pagine.
(slide)

2 Algoritmi di sostituzione delle pagine - pagina vittima

nella page table viene mantenuto un bit di validità a seconda che sia caricata o meno in ram, affianco al bit si trova un dirty bit che segnala quali pagine sono state modificate e accedute in scrittura. Se scelgo pagina vittima modificata devo ricopiarla su disco, altrimenti non devo ricopiarla in memoria secondaria.
è un problema comune nell'ambito dei sistemi operativi: sia memoria cache che altro, un problema ricorrente.
Obiettivo è quello che minizza i page fault. Un metodo per provare la bontà è provarli su una sequenza prefissata di accesso
Data una memoria fisica, se andiamo ad ampliare la memoria ci aspettiamo che il numero di page fault cali, all' aumentare di numeri di frame in memoria.

2.1 Algoritmo FIFO

rimpiazzo la pagina che da più tempo e in memoria principale.
soffre di anomalia di belady

2.2 Algoritmo ottimale

si rimpiazza la pagina che non verrà usata per il periodo più lungo.
Il problema di questo algoritmo è che non è implementabile, perché non si può sapere in anticipo quali pagine verranno utilizzate. è un algoritmo ideale, usato come punto di confronto per altri algoritmi praticabili.

2.3 Algoritmo Least Recently Used LRU

Algoritmo che studia il passato, rimpiazzo la pagina che non è stata usata per il periodo più lungo.
LOCALITà DEL PROCESSO: set di pagine di riferite, ipotizzo che nell'immediato futuro il set di pagine riferite non sarà molto diverso.
Non soffre dell'anomalia di belady, ampliando il numero di pagine i page fault non aumentano.

Tuttavia necessitano di un hardware complesso, per tenere traccia di tutte le pagine riferite.
Vengono implementate delle varianti ottenibili a livello software.

2.4 algoritmi di stack

se per ogni reference string r, per ogni memoria $M(m, r)$

2.5 implementazione LRU

abbiamo bisogno di assistenza hardware, abbiamo bisogno di contatori (in MMU) che viene incrementato ad ogni accesso in memoria. Nella page table manteniamo un registro che contiene il tempo di ultimo riferimento. Ogni volta che una pagina viene riferita vado a incrementare il (...) Pagina riferita più lontana nel tempo avrà il reference time più basso.
Nella page table per ogni entry un registro che contiene il tempo di ultimo riferimento (accesso). In più contatore MMU. Ogni volta vado a cercare processo riferito più indietro (basso). TUttavia è una ricerca, quindi costoso.
Presenta di lista ordinata mediante double linked, ogni volta che una pagina viene riferita viene spostata in testa alla lista.
Ogni volta pagina da liberare sarà quella in fondo alla lista. Può essere implementato in microcodice, ma costoso.
Quindi si implementano approssimazioni delle lru, che si basano su informazioni (non tempo di accesso) che possono essere utili per stimare quali verranno riferite in futuro.

2.6 Algoritmo Not Recently Used NRU

Si basa sul reference bit. il reference bit inizialmente è settato a 0, quando si fa riferimento alla pagina il bit viene settato a 1.
Pertanto ci fornisce informazione sul fatto che sia stata riferita o meno la pagina.
UN possibile algoritmo potrebbe cercare una pagina con reference bit uguale a 0, quindi maggiore probabilità che non venga riferita nell'immediato futuro.
è una variante molto grezza, imprecisa. SI preferiscono altre varianti.

2.7 Algoritmo Not Frequently Used NFU

ad ogni pagina si associa un contatore, a intervalli regolari si somma il reference bit al contatore e si resetta il bit.
il contatore aumenta soltanto se la pagina è stata riferita nell'ultimo intervallo di tempo.La pagina che hanno contatore più alto sono quelle che sono state riferite più frequentemente.
Difetto principale, pagina riferita nell'ultimo tick e pagina riferita molti tick indietro hanno lo stesso valore nel contatore, non tengo conto di quando sono

state riferite, ma soltanto quante volte.
Meglio tener conto del tempo di riferimento.

2.8 Algoritmo aging

Uso del bit di riferimento e sequenza di bit supplementari, per ogni pagina. (slide) a ogni tick si shiftano i bit a destra, e viene copiato il reference bit nel bit più significativo. Riazzero reference bit. Continuo così.Se devo eliminare pagina scelgo a quella con il numero più basso. Per essere precisi: intervallo di tempo più piccolo e avere più bit da destinare alla raccolta.

2.9 Algoritmo a orologio CLOCK - second chance

In questo caso le pagine sono organizzate in una lista circolare (slide), di volta in volta un puntatore al prossimo frame. Se trovo reference a 1, lo setto a 0 e procedo (concedo seconda chance alla pagina con reference 1). Se trovo pagina reference a 0, elimino questa pagina, porto quella nuova in memoria settando il reference a 1.
Buona approssimazione della LRU, usato in molti sistemi.

2.10 CLOCK migliorato

ovvero uso del dirty bit, ovvero so se pagina è stata modificata o meno.
Tra le pagine riferite meglio eliminare quella non modificata, perché posso rimpiazzare senza ricopiarla. Nel classificare le pagine da eliminare, la migliore è quella con entrambi i bit a 0.
ordine di scelta:

- non usata recentemente e non modificata, r=0, d=0
- non usata recentemente e modificata, r=0, d=1
- usata recentemente e non modificata, r=1, d=0
- usata recentemente e modificata, r=1, d=1

Primo giro con entrambi i bit a 0, senza modificare bit. AL secondo giro vado a cercare pagine con r=0 e d=1, azzerando i reference bit. Se non trovo, ripeto il primo giro. (slide)

3 Trashing

Fenomeno che avviene quando un processo non ha abbastanza pagine caricate in memoria principale, quindi si teme che si vadano a generare molte page fault.

Questo comporta a sottouso della cpu (basso uso) perchè continuo scarico e carico di pagine, processi impegnati in I/O. Questo potrebbe portare il sistemare ad aumentare il numero di processi, perché cpu sottoutilizzata, tuttavia questo peggiorerebbe la situazione in quanto i frame disponibili sarebbero ancora minori, peggiorando il TRASHING.
Bisogna monitorarlo per evitare il peggioramento. Trashing quando al processo sono assegnati troppi pochi frame rispetto alla sua richiesta e la sua LOCALITA ovvero numero di pagine logiche contemporaneamente utilizzate in un certo intervallo di tempo. Certo insieme limitato di pagine (slide).
Se processo ha numero di frame minore di LOCALITA, allora si ha trashing, continuo swapping e page fault. Avviene in particolare con algoritmo in cui la pagina vittima scelta è una delle pagine del processo stesso.

3.1 modello di working-set

numero di riferimenti a pagine che il processo faccia in un intervallo di tempo (esempio finestra di 10000 istruzioni). La finestra deve essere abbastanza larga in modo che il working set copra l'intera località. Se il delta è troppo grande copre più località. Delta dev'essere proporzionato alla dimensione delle località.

3.2 algoritmi di allocazione

Ci sono algoritmi di allocazione che si basano sullo studio del working set, il sistema tiene monitorizzato il working set di ogni processo e si occupa di allocare frame a sufficienza, un processo viene messo in coda ready soltanto se c'è abbastanza spazio in memoria. nel caso opposto (troppi pochi frame) alcuni processi vengono sospsesi.
Come approssimo i woeking set?