Working Set

Ne manteniamo uno per processo.
Stabilito un Δ, che indica il numero di ultimi accessi in RAM, il working set è l'insieme delle pagine usate negli ultimi Δ accessi.
Il working set si comporta come un'approssimazione della località
La scelta di Δ è fondamentale
- troppo: eccede
- poco: meno della località

Il working set non ha conflitti

A cosa serve? A capire un numero più preciso e giusto di frame da allegare a un processo
Permette dunque la prevenzione del thrashing più individuale che globale.
$D = \sum WSS_i$ (size del WS_i) Permette il calcolo globale di frame che servono
D più grande della memoria disponibile → thrashing
D più piccolo → situazione più stabile

Calcolo del WS nella pratica (quanti strumenti permettono un'approssimazione)
- interrupt periodici
- bit R
- log con la storia di R basta su Δ

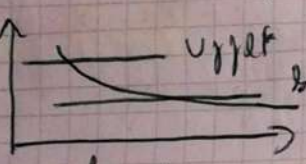
Si potrebbe velocizzare l'avvio di un programma con la tecnica del working set model.
Un programma all'avvio tende sempre ad avere molti fault non avendo pagine in memoria ma potremmo far conoscere al SO quali sono le pagine che vengono usate all'avvio per velocizzarlo, dunque
- Working set model: working set dell'avvio di un programma
- prepaginazione: caricare pagine (quelle del WSM) ancora prima dell'esecuzione del programma.

Page Fault Frequency

Modello più diretto per approcciare il thrashing

NB: secondo il modello teorico, a meno della seconda di Belady, più frame ci sono e meno fault ci saranno

Monitoriamo la PFF del processo in relazione al numero di frame a lui dedicati



Definiamo un upper e lower bound che creano un range di accettabilità

Se mi trovo sopra al upper faccio delle operazioni per aiutare il processo (è in thrashing)
Infatti il suo WS sarebbe più grosso delle frame necessarie.
Se sto sotto al lower bound mi rendo conto che al processo sta venendo dedicato un numero di frame più grosso del WS, e ci posso stare altri processi in training potrei prendere frame da questo processo per dedicarle a loro e per trovare un equilibrio
Sistema in sovraccarico: tutti sopra al upper

Grafico che mostra il comportamento del rate nel tempo.

I picchi sono dovuti al passaggio da una località all'altra. All'inizio subito dopo la transizione aumentano i fault. I picchi dunque ci fanno capire che c'è stato un passaggio di località!

Pulitura

Il meccanismo di gestione page fault diventa più efficiente se ci sono frame liberi disponibili poiché non dovremo applicare l'algoritmo di swapping.

Page daemon: processo di sistema che si occupa di tenere, quando può, una pool di frame liberi. Si occupa di ~~fare~~ anticipare la scelta della eventuale pagina da buttare

- Seleziona, libera e in caso pulisce la pagina

Sostanzialmente stiamo anticipando l'esecuzione dell'algoritmo di scelta in momenti in cui la CPU non è particolarmente occupata

Caso sfortunato: il daemon ha scelto una pagina che sta per essere usata

In realtà è abbastanza gestibile senza I/O grazie al rifacciamo della pool dei frame liberi

La pulitura è comune sia su Linux che Windows

Dimensione della pagina

Bisogna trovare un buon compromesso per la scelta della dimensione della pagina

Pagina grande:
- tabella delle pagine più piccola
- miglior trasferimento I/O (avremo più contiguità su disco delle word)
- minimizza i page fault

Pagina piccola:
- minore frammentazione interna
- WS definito meglio: l'approssimazione diventa più precisa (meno sprechi)

Ha una certa relazione con la dimensione del blocco su disco (di solito è un suo multiplo)

Pagine condivise

I processi possono condividere in vari modi alcune pagine (es: le pagine del codice)
Bisogna fare attenzione se sono pagine di
- sola lettura: condivisione semplice
- lettura/scrittura: si fa IPC con memoria condivisa

Il modello delle pagine condivise è implementato sulle tabelle classiche e multilivello in modo semplice ed efficiente

Difficoltà:

- Cache: Se abbiamo una cache basata su indirizzi virtuali potremmo avere problemi di aliasing (la stessa frame fisica potrebbe avere nomi differenti nella memoria virtuale dei processi che la condividono)

Il problema sorge anche usando gli ASID: Usando gli ASID potrei avere linee di cache diverse che hanno però riferimento alla stessa word fisica

Soluzioni:
- disattivare caching per pagine condivise
- fare ricerca in cache basata su ind. v. e tag fisici

Nella cache ogni linea assume come chiave (ASID, C.V.) Si fa la ricerca nella cache e si trovano dei candidati (col tag fisico), nel mentre l'MMU fa ricerca nella TLB e vede se c'è una corrispondenza

Se c'è corrispondenza tra i candidati con l'i.f. dato in output dalla TLB ho trovato dei duplicati

Dunque la L1 e l'MMU collaborano in parallelo e trovano i duplicati

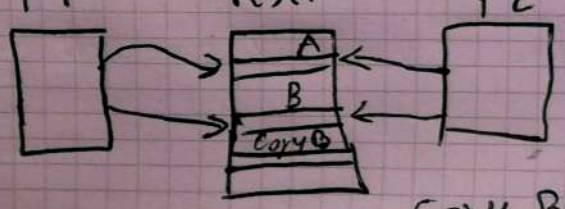
Tabella inverted delle pagine

Nei single core: dovremmo fare un'alterazione della tabella in caso di context switch o larga parte. Il SO sa tutte le etichette che ha una pagina, perciò riesce a risolvere i problemi di aliasing. Dunque in caso si verifica un evento per cui si cerca un c.v. nella tabella e non c'è più in realtà l'indirizzo fisico a cui la riferimento (con un altro c.v.) il SO risolve il problema con un'alterazione della tabella (cambiare l'etichetta). La struttura è modificata a favore dell'ultimo c.v. inserito e così evitiamo l'I/O

Nei sistemi multi core: la cosa diventa praticamente ingestibile. L'aliasing non è gestibile efficientemente.
L'unica "soluzione" sarebbe avere delle tecniche tabelle con corrispondenza molti-a-uno

Copy-On-Write

È una possibile ottimizzazione per la gestione della condivisione pagine
Idea: condividere in lettura/scrittura una pagina (con dei dati in comune) anche senza la esplicita richiesta dei processi
Se uno dei due processi tenta di scrivere la pagina condivisa, essa è copiata in un altro frame (con la modifica).
Il secondo processo rimane con la vecchia pagina che non ha subito modifiche



P1 scrive su B, allora si crea Copy B, a questo punto la modifica vera e propria viene su Copy B. P1 punterà su Copy B e P2 su B

Sostanzialmente questo meccanismo risparmia RAM (almeno temporaneamente)

Nella pratica le fasi combine hanno il privilegio di scrittura da P1 e P2, il SO nel caso in cui fatta richiesta di scrittura se ne accorge e fanno il meccanismo copy-on-write

Dopo il meccanismo P1 avrà i genitori WR su copy B e P2 avrà i genitori WR su B.