Lezione 12/05

A parità di algoritmo, se aumento i frame a disposizione, mi aspetto che ci sarà un numero di page fault migliore. che diminuisce.

Anomali di Belady: aumenta il numero di frame, e aumentano i page fault. Vedi esempio slide, dove nel passaggio da 3 a 4 frame ci sono più page fault. Perché si verifica? Perché in realtà è sbagliato pensare che bisognerebbe rimuovere il frame che c'è da più tempo, non è un buon criterio.

$B_t(n)$ indica un set di pagine caricate in memoria in un determinato istante t considerando fissato un determinato algoritmo, con una data sequenza, la proprietà di inclusione vale quando l'insieme delle pagine che terrei in memoria all'istante t avendo n frame, è un sottoinsieme con un frame in più nello stesso istante. Questa proprietà non vale sempre. Es. FIFO.

LRU la soddista: in un dato istante, a tempo t, controlla le n pagine che ha in memoria quando sono state referenziate per l'ultima volta. Se ho a disposizione un frame in più, allora l'insieme delle n pagine, più una, non può che essere un sottoinsieme di quella con n. Perché è come se riuscissimo a vedere "oltre", e a salvarne una in più. Chiaramente, quelle che salviamo con n pagine, saranno anche salvate nelle soluzioni con n+1 pagine, sennò non avremmo quelle n pagine, ma tra quelle ne avremo alcune diverse.

Anche l'algoritmo della seconda chance presenta l'anomalia, in quanto non è altro che un algoritmo simile a FIFO.

Allocazione dei frame

Problema satellite: come assegno i frame ai miei processi? Paginazione su richiesta (**pure demand paging**). Supponiamo di avere un processo appena avviato, e supponiamo che tutte le pagine del processo stiano su disco. Inizialmente, devo decidere io cosa portare in RAM, e quindi all'inizio non c'è nulla. Cosa devo portare in RAM? La risposta potrebbe anche essere quella di non portare niente. La paginazione su richiesta, non porta nulla in RAM. Inizialmente non carico nulla in RAM, e questo porta ovviamente al fatto che inizialmente verranno causati una marea di page fault. Quindi già il page fault verrà causato

non appena verrà richiesto il main, e così via per tutte le fasi primordiali di avvio del processo, es. Program Counter etc. Questa tecnica, se consideriamo la **page fault frequency,** porterà che a tempo 0 ci sarà un picco di page fault, e poi si vanno a stabilizzare. Se ho tanti page fault, questo porta overhead.

Se ho la mia RAM e un certo numero di processi, posso immaginare di fare una sorta di assegnamento.

Quanti frame posso assegnare a un processo? Dipende, volendo potrei anche assegnarglieli tutti, occupando la RAM.

Ma per eseguire un processo qualunque, ho un numero minimo di frame da assegnare? Numero minimo di frame affinché il processo sia eseguibile (non "eseguibile non troppo lentamente", ma eseguibile). Osserviamo che una specifica architettura ha un numero minimo di frame da assegnare per fare sì che ogni processo sia eseguibile.

Es. supponiamo di avere un add. Che significa esegurie una add? Questa istruzione sta in una pagina. Se la pagina non è in memoria, scatenerà un page fault. Supponiamo che al processo sia assegnato un solo frame. Ovviamente, se chiede altri frame, scatenerà dei page fault.

La fase di fetch dell'esecuzione chiede di andare nell'indirizzo di memoria referenziato. Si tratterà sicuramente di un'istruzione di memoria sicuramente in RAM. Il fetch dedicato al processo verrà destinato a contenere la pagina che contiene l'addendo in questione. Sulla stessa istruzione, verrà generata una seconda istruzione, non per il prelievo dell'operatore ma per il prelievo dell'operando. Così rieseguo l'istruzione, ma ho di nuovo un page fault, sul prelievo dell'istruzione. Il re-iterare la richiesta provocherà un secondo page fault. Quindi avrò infiniti page fault. Un solo frame non basta. Quindi ne servono due? Dipende dall'architettura, dalla generazione di CPU, etc.

Qual è il minimo? Devo garantire che nei miei frame ci sia l'istruzione. Dipende da quanto è grande l'istruzione. Se è così grossa da ricadere in più word? Già per ospitare un'istruzione in sè, devo prevedere due pagine. Se ho un operando che può stare in RAM, inoltre devo dedicare almeno un frame per questo. Se in realtà gli operandi che possono fare riferimento alla RAM sono due, allora in totale i frame sono 4. Ci sono altri meccanismi che possono aumentare questo riferimento strutturale. Prendo un indirizzo di memoria. Ma questo è una sorta di riferimento indiretto. Significa che la semantica di quell'operazione non sta in

quell'indirizzo. In realtà in quell'indirizzo ci sta una word, dove c'è l'indirizzo della mia operazione. Quindi poi bisogna fare un secondo accesso nella fase di fetch per accedere al vero operando. Quindi nell'ambito di una singola istruzione per gestire un solo operando io debba fare due accessi che potenzialmente si trovano in pagine distinte. La gestione di questi operandi ha bisogno di due frame ciascuno. Alcune architetture possono supportare livelli multipli di indirezione. Per eseguire una singola istruzione di un'architettura, non posso dire che un singolo frame va bene. Inoltre devo cercare di non avere un overhead troppo alto. Ma al di là di questo, è interessante capire che c'è un limite teorico sotto al quale non ha neppure senso cercare di eseguire il processo.

Quanti frame assegnaere?

Primo approccio: **allocazione equa.** Inoltre, bisogna considerare che il SO, al netto di quello che resta (visto che servono frame anche a lui), poi ripartisce quello che resta.

Vedremo che la quantità di RAM da assegnare è legata alle sue esigenze.

Allocazione proporzionale: processi con taglia piccola, intesi come codice e dati, avranno assegnazione più risicata in termini di pagine e di frame. Posso immaginare di suddividere la disponibilità globale di m frame, considerando la somma delle varie taglie, con s_i che indica la specifica taglia del processo, e così otteniamo il numero di frame da assegnare all'i-esimo processo, indicato con a_i ovviamente da approssimare per difetto.

Assegnando una certa quantità ad a_i, se a un certo punto aumentano i processi, qualcuno ne deve rimanere "fuori", cioè avere meno processi.

Quando uno di questi processi provoca page fault, uno dei processi deve prendere la frame e portarla in RAM. Regola: se P3 ha provocato il page fault, quando andrò ad applicare l'algoritmo di sostituzione delle pagine, quali pagine dovrò considerare per essere scartate? Ovviamente solo le pagine di P3. P3 ha 4 frame, e quindi deve cederne una per fare spazio per la sua ulteriore richiesta.

Sembra una scelta sensata, quasi l'unica possibile. Questo principio appena enunciato corrisponde in realtà all'allocazione locale. Esiste l'allocazione **globale:** quando P3 provoca un page fault, applico l'algoritmo di sostituzione delle pagine prescelto, e le pagine papabili, sono tutte le pagine presenti in RAM. Bisognerà quindi riferirsi a tutti i processi. Ma quindi per un problema di P3 ci vanno di

mezzo gli altri processi? È un modo furbo di gestire le risorse che abbiamo a disposizione. Se c'è un processo che ospita 5 pagine, ma ha la pagina che da più tempo non viene utilizzata, allora forse quel processo, paragonato a tutti gli altri alle esigenze che sembrano manifestare, magari è quello che "merita" che gli venga tolta la pagina. Modo ponderato che tiene conto delle esigenze di tutto il sistema.

E che succede al processo che è proprietario del frame scelto? Gli viene decurtato il frame dal processo che ha causato il page fault, e quindi questo avrà un frame in meno rispetto al numero di frame assegnati all'inizio, mentre il processo che ha causato il page fault ne avrà uno in più, e ciò rispecchia le proprie esigenze. Inoltre, il processo "vittima" potrebbe anche non accorgersene. Se le sue esigenze rimangono le stesse, i frame che gli restano gli bastano senza problemi. **Vantaggio:** scelte fatte a priori, senza fare distinzioni, e inoltre quello di non dover fare partizioni, e fare selezioni dinamicamente.

E se il processo scende verso quel limite strutturale? Il processo potrebbe essere sospeso o fare swap per ottenere ulteriori frame. Il minimo strutturale però è molto teorico, quindi in generale consideriamo che abbia "pochi frame". Potrei avere situazione di **thrashing:** ho un alto numero di page fault rispetto alla normale soglia che mi aspetto. Si parla di thrashing quando il numero di frame diminuisce molto e l'overhead supera circa il 50% dell'esecuzione del processo. Il processo sta lavorando male. Quando la memoria è scarsa, la situazione è probabile che peggiori esponenzialmente e che poi io mi ritrovi un processo bloccato. Può diventare un problema se sto applicando una strategia locale. Se ha troppi pochi frame per le sue esigenze, allora il processo soffrirà, dato che l'allocazione locale non permette di acquisire o perdere frame. Potrei avere quindi un processo che lavora male. Problema localizzato su quel processo. Se riuscissi ad identificare il processo sofferente, sempre supponendo di fare allocazioni rigide, aggiustare il tiro e concedergli più frame. L'allocazione globale invece "ruba" frame che ne hanno più in abbondanza in base alle loro esigenze, e li dona a chi ne ha più bisogno. Questo permette di risolvere i casi di singolo trash. E che succede se tolgo frame a uno per poi creare sofferenza da un'altra parte? Se il thrashing passa da un processo all'altro? Magari ci sono troppi processi o comunque ho poche risorse per i processi presenti. Ho un sistema sovraccaricato.

Cosa posso fare per evitare di ritrovarmi in situazioni sofferenti come questa e andare incontro alle esigenze dei miei processi?

Modello di località: insieme di word/informazioni che sono utili per una data frazione di esecuzione del nostro processo, e ciò può riguardare porzioni di codice e dati. Riguarda un dato processo in un dato momento. A parità di processo in momenti diversi, la località può cambiare nel tempo. Se garantisco a ogni processo di poter ospitare tutte le parti della sua località attuale, allora posso essere abbastanza sicuro che quel processo potrà lavorare bene. Esigenza deve essere coperta in termini di pagine. Se, per esempio, gliene do la metà, il processo provocherà molti page fault. Se gliene assegno di più, continua a lavorare bene, ma forse è una sorta di spreco. Tentativo di misurare codice e dati in base alla località del processo. Se potessi monitorare la località dei miei processi in un dato istante potrei gestire meglio l'allocazione dei frame. Chiaramente la località cambia. Può cambiare anche la quantità di dati che rappresenta la mia località. Si può passare da località piccole a località più grandi.
Come tracciare la località? Ci sono degli esperimenti. **Working set:** stabilito un certo numero delta di accessi, posso definire come working set, in base a un certo tempo t, il working set di quell'istante. Quell'insieme di pagine che sono state referenziate negli ultima delta accessi alla memoria. A me non interessa l'indirizzo virtuale ma la pagina in cui ricade. Devo ragionare su una sequenza di indirizzi virtuali, quindi la sequenza di pagine referenziate. Trasformiamo la nostra sequenza in una sequenza di pagine referenziate. Se delta corrisponde a 10 accessi, il working set, a parità di processo, muta. Se ci sono delle ripetizioni, queste hanno un impatto ben preciso. Working set può variare sia internamente che in termini di dimensioni. Posso puntare ad avere una definizione di working set che copra la località, la arrotondi. Come se volessi approssimarla in termini di pixel. Per tracciare il working set, potrei implementare alcune tecniche: potrei misurare la dimensione del working set di tutti i mei processi in un dato istante, cercando di capire se sono in una situazione in cui sarò in sofferenza perché devo fare cose oltre alla mia possibilità effettiva, e vedere se posso coprire queste esigenze in qualche modo. Ciò permette di gestire i problemi di caching. Si manifesta quando la quantità di frame assegnata a un processo è sproporzionata, ridotta, rispetto alle esigenze concrete. Se ne ho assegnate troppo rispetto alle esigenze concrete, posso, in una strategia di allocazione locale, aggiustare queste assegnazioni puntualmente.

Fare un tracciamento del genere, in cui vedo quali pagine sono state usate negli ultimi delta accessi, è possibile? Via software, dovrei segnare ogni volta la pagina

usata, e quindi sarebbe pesante da fare. Potremmo però fare una sorta di approssimazione: periodicamente, prima di azzerare i bit di referenziamento, potrei in qualche modo tracciarli, ricordarmene, e notare che questo tipo di lavoro lo facciamo già. La definizione stessa di working set si basa sul numero di accessi, ma non ho questa informazione. Devo immaginare un contatore per ogni frame, e nella parte alta posso vedere quali sono le pagine che sono state usate negli ultimi quattro cicli. Se delta copre un milione di istruzioni, e so che ogni ciclo elaboro 500000 istruzioni, magari devo guardare due cicli, circa. Magari non posso dire esattamente quanti accessi faccio, ma posso vedere i bit di referenziamento nelle parti più significative.

Esiste un modo più dinamico e che guarda al problema da un altro punto di vista, partendo dalla page fault frequency. Se la PFF è bassa, prossima allo 0, non significa per forza che io stia facendo un buon lavoro, magari ho esagerato col numero di frame per un processo e ho un processo che invece sta "soffrendo". Potrei mettere in correlazione la PFF con la effettiva quantità di frame che dovrei assegnare. Mette in correlazione il numero di frame con i page fault attesi. Se la PFF è troppo bassa forse sto esagerando. Idea: potrei creare una sorta di soglia di allarme superiore, dove capisco se il mio processo sta lavorando male. Es. se ha troppo lavoro da fare in base ai frame che gli ho dato. Anche se scende sotto una certa soglia, significa che sta lavorando comodamente, ma se c'è un processo con una PFF molto più alta, probabilmente dovrei prendere un processo da questa zona più comoda e darlo a chi sta in una zona più scomoda.

Quando faccio un cambiamento di località, e quindi di working set, passo da un working set a un altro. Mutano le mie esigenze ma anche le pagine di cui ho bisogno. Si possono creare delle transizioni. A volte può essere più graduale, fra due working set successivi possono esserci delle intersezioni etc. Fondamentalmente, il numero di page fault si alza, e se prima lavoravo su un certo numero di pagine, fa scattare un numero superiore rispetto alla norma di page fault. La PFF torna ad abbassarsi per poi vedere un altro picco nel momento in cui mi sposto su un working set diverso. Posso fare inferenze guardando la PFF su aspetti che apparentemente sono scorrelati. Ma in realtà lavorano sugli stessi aspetti.

A parità di stato di referenziamento, viene preferita una pagina pulita. C'è una routine, una sorta di demone, **paging daemon,** che si occupa di fare pulizia, per garantire che ci siano dei frame liberi.

L’uso dell’algoritmo di sostituzione delle pagine costa. Non troppo, ma costo. Un aggravo di costi si verifica se poi la pagina scelta è sporca, quindi devo pulirla. Per questo cerco di garantirmi un certo pool di frame liberi, per evitare page fault. Se ho un grado di multiprogrammazione non banale a un certo punto i processi tenderanno a espandersi e occuperanno tutta la RAM. Quindi tutti i frame vengono occupati. Il paging daemon vuole garantire questa sorta di riserva. Come alimento questo pool di frame liberi? Senza creare disservizi, ovviamente. Comunque sono frame che verranno rimossi durante l’algoritmo di sostituzione delle pagine, quindi lo “anticipiamo”. Facciamo partire il paging daemon periodicamente. Magari quando il SO ha poche cose da fare. Monitora pool di frame liberi, e si attiva per cercare di ripopolarlo, per aggiungere frame liberi, selezionando una pagina vittima, e in linea teorica “liberandola”, quindi sottraendola al proprietario e mandandola nel pool. In assenza di questa azione il processo avrebbe continuato a usare questa pagina, e magari ne aveva bisogno. Sto penalizzando i miei processi in nome di una futura esigenza? Solo apparentemente è vero. Istanza di algoritmo di sostituzione applicato in modo globale. L’algoritmo, se funziona bene, significa che il processo non se ne accorge e non viene penalizzato in termini di overhead e page fault. Ma l’algoritmo potrebbe anche sbagliarsi, e ha tolto una pagina che poteva essere una buona idea non togliere. Penalizzazione che sulla carta era una scelta vincente. Quando vado a selezionare una pagina che viene sottratta al processo e viene messa nel pool di pagine pulite, quest’operazione è proattiva: l’avrei comunque fatta, ma sto solo anticipando quando l’algoritmo viene applicato e quando avviene la pulizia (sincronizzazione tra memoria e disco). In assenza della tecnica del paging daemon, lo avrei dovuto fare dopo: quindi invocazione dell’algoritmo di sostituzione delle pagine e doppia operazione (scrittura e lettura). E questo scatta in modo sincrono sul page fault. Il processo che lo ha causato dovrà aspettare di più se è sfortunato. Ottimizzazione: il processo si ritroverà con un pool di frame liberi dopo il page fault. Esiste la possibilità, anche se minima, che quella rappresenti un passo falso. Tuttavia, consideriamo che la pagina, se era sporca, l’ho sincronizzata. Inoltre, se mi sono sbagliato e il processo proprietario mi reclama quella pagina, bisogna considerare che quella pagina è ancora in RAM. Non è page fault vero, perché sennò dovrei riprenderla dal disco e portarla in RAM. I frame che ne fanno parte, del poll, mantengono il vecchio contenuto. Page fault che non richiede nessun input/output. Una sorta di page fault veloce, grazie al **ripescaggio.** Ho anticipato la scelta di sostituzione delle pagine e ho pulito la pagina. In quasi tutte le

situazioni ho un vantaggio. Il bit di presenza/assenza viene posto a 0. Ma il SO si rende conto che quella specifica pagina di quello specifico processo è ancora fra i frame liberi.

Dimensione della pagina: sceglierla più o meno grande, cosa comporta? Dal punto di vista pratico, con quali vincoli? Qualcosa da gestire tra hw e sw, quindi MMU motore principale, e l'hw pone i suoi limiti. Nella pratica ha un range di dimensioni supportate. Va da dimensioni di 4KB fino a 8, 16, 32 etc, sempre potenze di 2. Muovendoci in questo range, il SO può fare un tuning su quello che questa scelta comporta. Se pensiamo di aumentare questo parametro, e quindi avere pagine più grandi, che vantaggio abbiamo? Ho una voce per ogni pagina virtuale. A parità di virtual memory, usare pagine più grosse mi permette di avere meno pagine virtuali e tabelle più piccole. Così ho meno voci. Pagine più grosse, meno voci, tabella invertita più piccola. Meno ingombrante. Da un certo punto di vista, usare pagine più grande permette di ottimizzare il comparto di I/O per la gestione delle pagine. Se ho una richiesta di lettura di 10 blocchi, e questi si trovano uno di seguito all'altro, il tempo di posizionamento lo pago una volta sola. E così le informazioni si trovano sotto la struttura del disco e non pagherò il tempo di posizionamneto. E quindi le letture sono una dopo l'altra. Caso peggiore: letture sparse, anche su "sezioni" diverse. Ma se posiziono bene i miei dati, posso rendere più probabile il caso migliore. Quindi, se le informazioni sono rappresentate su pagine grosse, es. 16KB coperta da una sola pagina, anzichè 8 pagine da 2K o 4 da 4K, il fatto di lavorare su pagine grosse mi assicura quella contiguità su disco. In linea di principio, se uso pagine più grosse, riduco il numero di page fault provocati, perché se io ho pagine piccolissime, ogni volta che le vado a trasferire, ho tanti più page fault che guardano a entità più piccole. Ma il page fault si paga indipendentemente da quanto sono grandi le pagine. In linea di principio, diminuendo la grandezza delle pagine, potrei aumentare il numero di page fault. Questo porta a uno scenario limite. Tutto si riduce a un caso teorico limite: tutto il processo contenuto in una sola pagina (ragionamento teorico/asintotico). Quindi lavorare con pagine grosse mi porta a ridurre il numero di eventi di page fault. Viceversa, se uso pagine piccole potrei avere altri vantaggi: minore frammentazione interna. Se quella pagina è troppo grande, magari rischio di avere uno spreco, e quindi avere in RAM informazioni non richieste e non necessarie. La mia granularità mi permette di lavorare meglio e quindi non sprecare RAM. Collegamento sul modo in cui i nostri dischi lavorano: i nostri dischi

hanno una dimensione di base sui blocchi supportati su qualsiasi operazione di I/O. Cioè, ogni disco ha un blocco di base. Qualunque lettura avviene per blocchi. Obbligati a fare operazioni di lettura/scrittura. [da rivedere].

Pagine condivise

Ci sono situazioni in cui processi distinti o magari imparentati possono avere contenuti in comune. Il caso più semplice di pagine condivise è ad esempio lo scenario in cui ho processi distinti che lavorano con lo stesso binario, lo stesso testo. Ho istanze multiple dello stesso programma. Es. supponiamo di avere un editor. Magari devo editare tre documenti diversi. Immaginiamo il blocco note. Abbiamo la virtual memory organizzata con stack, vuoto, heap, code. La zona di code è uguale per tutti i processi di questo tipo, quindi potrei pensare di salvare una sola copia del codice in RAM. Tutti i processi condividono quindi il contenuto del frame relativo al segmento code. La zona dei dati, invece, è ovviamente non condivisa. Processi lavorano come se ognuno avesse la propria copia, e quindi è solo conveniente. Per renderlo possibile, questa zona deve essere in sola lettura. **Codice rientrante:** codice non ha bisogno di auto-modificarsi. Se invece uno dei processi stesse lavorando in tempo reale sul codice, questo porterebbe a modifiche visualizzabili anche agli altri processi, e non è quello che vogliamo. Non sarebbe una vera ottimizzazione. Se invece metto un READ-ONLY su queste zone, non si verificano problemi. E se c'è bisogno di modificare qualcosa? Processi distinti richiedono al SO di mappare una parte di memoria. Processi isolati condividono solo una parte del segmento. Come lo si implementa? La pagina in questione dovrà essere referenziata dai processi che vogliono scrivere sul codice. Es. il processo richiede di scrivere esplicitamente. Magari ci sono processi che vogliono condividere la pagina, e vogliono cooperare, e quindi anche se entrambi sono in scrittura puntano alla stessa pagina.

Problema: tabelle invertite. Uno dei motivi per cui non si usano granchè. Ricordiamo che cerco la voce Processing_id: frame. Se trovo la coppia cercata, ottimo, sennò page fault. Problema: se più processi condividono un frame, che processing_id metto? La soluzione è quella di fare una ricerca iniziale, e se non trovo il processing_id cercato, modifico il processing_id assegnato al frame cercato, e rieseguo l'istruzione, e questo scatena nuovamente una ricerca. Però questo è overhead. Problemi soprattutto su sistemi multi-core.

Gestione della cache: processo P1 potrebbe vedere la pagine come pagina virtuale 100 e P2 come pagina virtuale 1000. Problema dell'aliasing. Lo stesso contenuto ha due alias, due nomi. P1→ pagina 100, P2→ pagina 1000. Problema con cache basata su indirizzi virtuali.

Non è uno spreco, è un problema di coerenza. Il problema è che P1 cerca informazione 100 nella cache, non la trova, e va in cache. P2 fa la stessa cosa. Abbiamo la stessa informazione su due alias diversi. Non è solo un problema di spreco o di overhead, ma di **coerenza.** Se P1 modifica quel segmento, la modifica resta in cache. Le linee di cache ovviamente non sono syncate. Se avessimo la cache fra MMU e RAM e non fra CPU e MMU, risolveremmo il problema. La soluzione però non è abbandonare la cache frapposta fra CPU e MMU. L'idea è che quando faccio una ricerca e non trovo riscontro all'interno della cache, non è detto che sia un cache miss. Io devo aspettare che l'MMU tiri fuori l'indirizzo fisico. Una volta fatto, avrò un riscontro se questo è presente nelle linee di cache o meno. Ma io una parte già ce l'ho, ho l'offset, che è la parte bassa. La parte bassa rimane invariata. Se ho linee di cache da 64 byte, gli ultimi 4 bit sono quelli che individuano le informazioni all'interno di quei 64 byte. Gli ultimi 4 bit non mi interessano, mi interessano gli altri, che distinguono una linea di cache dagli altri. Se non ho occorrenze trovate, allora non c'è l'informazione richiesta nella cache. Se invece trovo occorrenze, aspetto la MMU, velocizzata pure dalla TLB. Tenendo conto che [...].