

Sistemi Operativi Modulo I

Primo canale (A-L) e Teledidattica

A.A. 2021/2022

Corso di Laurea in Informatica

La Gestione della Memoria

Igor Melatti

Sapienza Università di Roma
Dipartimento di Informatica

Roadmap

- Gestione della memoria: requisiti di base
- Partizionamento della memoria
- Paginazione e segmentazione
- Memoria virtuale: hardware e strutture di controllo
- Memoria virtuale e sistema operativo
- Gestione della memoria in Linux

Perché Gestire la Memoria (nel SO)

- La memoria è oggi a basso costo, e con trend in diminuzione
- Tuttavia, ciò è più che bilanciato dal fatto che le moderne applicazioni richiedono sempre maggiore memoria
- Se si lasciasse a ciascun processo la gestione della propria memoria, ogni processo userebbe semplicemente tutta la memoria a disposizione
- Niente più multiprogrammazione, che abbiamo visto essere essenziale
- Si potrebbe allora imporre dei limiti di memoria a ciascun processo
- Diventa però difficile per un programmatore scrivere un processo che rispetti tali limiti
 - anche se, agli inizi, era proprio così

Perché Gestire la Memoria (nel SO)

- Quindi, ci pensa il sistema operativo
- I processi fanno finta di avere tutta la memoria a loro disposizione, e sta al Sistema Operativo rendere realistica questa illusione
- Gestire la memoria include lo swap di blocchi di dati dalla memoria secondaria
- Questa gestione di I/O è ovviamente più lenta del processore
 - il SO deve pianificare lo swap in modo intelligente, così da massimizzare l'efficienza del processore
- Occorre gestire la memoria affinché ci siano sempre un numero ragionevole di processi pronti all'esecuzione, così da non lasciare inoperoso il processore

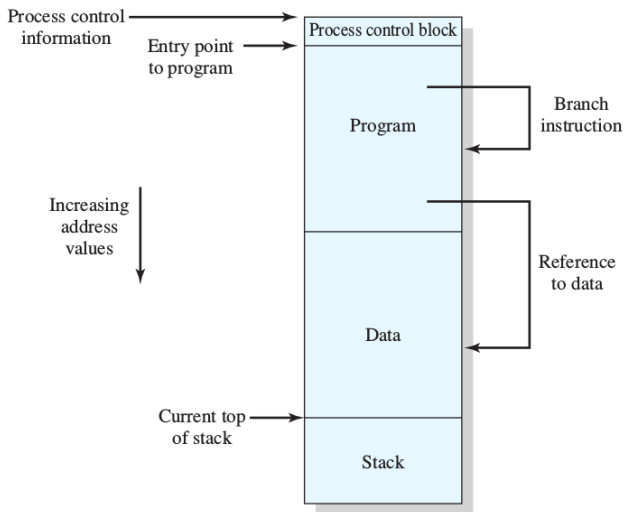
Requisiti per la Gestione della Memoria

- Rilocazione
 - importante che ci sia aiuto hardware
 - *aiuto*, non gestione diretta: sistema operativo ed hardware collaborano
- Protezione
 - importante che ci sia aiuto hardware
- Condivisione
- Organizzazione logica
- Organizzazione fisica

Requisiti: Rilocazione

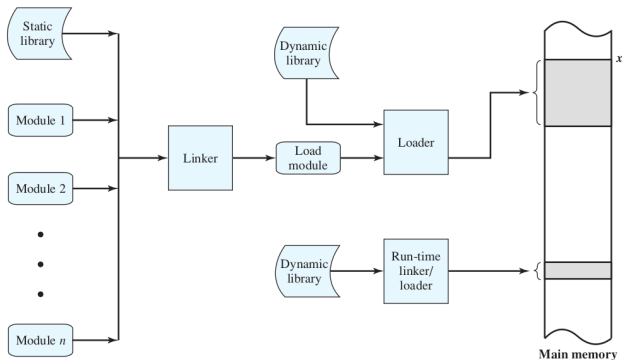
- Il programmatore non sa e non deve sapere in quale zona della memoria il programma verrà caricato
 - potrebbe essere swappato su disco, e al ritorno in memoria principale potrebbe essere in un'altra posizione
 - potrebbe anche non essere contiguo, oppure con alcune pagine in RAM e altre su disco
 - in questo contesto, per “programmatore” si intende o chi usa l'assembler o il compilatore
- I riferimenti alla memoria devono essere tradotti nell'indirizzo fisico “vero”
 - preprocessing o run-time
 - se run-time, occorre supporto hardware

Rilocazione: gli Indirizzi nei Programmi



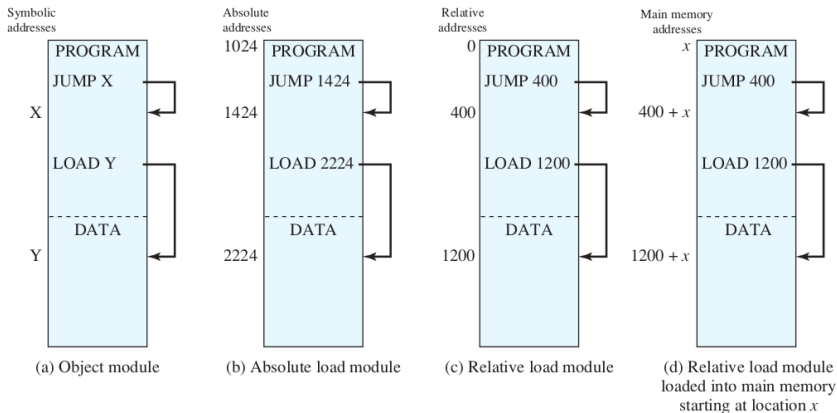
Rilocazione: gli Indirizzi nei Programmi

- Generazione di codice eseguibile: il *linker* (collegatore) mette tutto insieme, tranne le librerie dinamiche
 - il risultato è un *load module*, perché può essere trattato dal *loader* (caricatore, sott. in memoria principale)



Rilocazione: gli Indirizzi nei Programmi

Nota bene: (b) prevede la sostituzione degli indirizzi nel programma; (c) e (d) invece sono a run-time



Logici: il riferimento in memoria è indipendente dall'attuale posizionamento del programma in memoria

Relativi: il riferimento è espresso come uno spiazzamento rispetto ad un qualche punto noto

- caso particolare degli indirizzi logici

Fisici o Assoluti: il riferimento effettivo alla memoria

Rilocazione

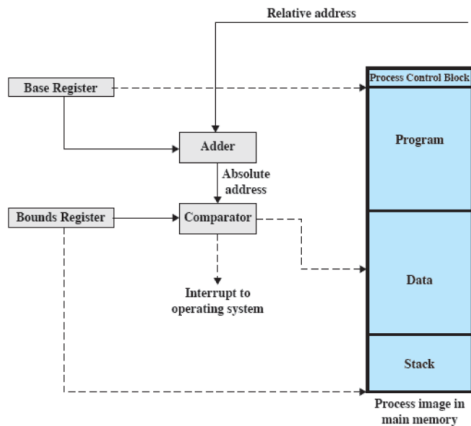
- Vecchissima soluzione (CTSS): gli indirizzi assoluti vengono determinati nel momento in cui il programma viene caricato (nuovamente o per la prima volta) in memoria
 - si può fare senza hardware dedicato
 - nel CTSS, reso ancora più facile: 1 programma per volta, sempre a partire dall'indirizzo 5000; si può fare (b)
- Soluzione più recente: gli indirizzi assoluti vengono determinati nel momento in cui si fa un riferimento alla memoria
 - serve hardware dedicato
- Come detto, un programma potrebbe essere messo in diverse partizioni, a causa di swapping e compattazione

Rilocazione a Run-Time senza Hardware Speciale

- Ogni volta che un processo viene riportato in memoria, potrebbe essere in un posto diverso
- Nel frattempo, potrebbero essere arrivati altri processi e averne preso il posto
- Quindi, ad ogni ricaricamento in RAM, occorre ispezionare tutto il codice sorgente del processo
- Sostituendo man mano tutti i riferimenti agli indirizzi
- Troppo overhead: i costruttori di hardware decidono di aiutare il SO
 - diventa proprio impossibile con la memoria virtuale: ci torneremo

Rilocazione a Run-Time con Hardware Speciale

Soluzione per indirizzi relativi



Registri Usati per la Rilocazione di Indirizzi Relativi

- Base register (registro base)
 - indirizzo di partenza del processo
- Bounds register (registro limite)
 - indirizzo di fine del processo
- I valori per questi registri vengono settati nel momento in cui il processo viene posizionato in memoria
 - mantenuti nel PCB del processo
 - fa parte del passo 6 per il process switch (vedere slides sui processi)
 - non vanno semplicemente ripristinati: occorre proprio modificarli

Registri Usati per la Rilocalizzazione di Indirizzi Relativi

- Il valore del registro base viene aggiunto al valore dell'indirizzo relativo per ottenere l'indirizzo assoluto
- Il risultato è confrontato con il registro limite
- Se va oltre, viene generato un interrupt per il sistema operativo
 - simile al segmentation fault...

Requisiti: Protezione

- I processi non devono poter accedere a locazioni di memoria di un altro processo, a meno che non siano autorizzati
- A causa della rilocazione, non si può fare a tempo di compilazione
- Quindi, bisogna farlo a tempo di esecuzione
- E pertanto, serve aiuto hardware

Requisiti: Condivisione

- Deve essere possibile permettere a più processi di accedere alla stessa zona di memoria
 - ovviamente, solo se è effettivamente utile allo scopo perseguito dai processi
- Caso tipico: più processi vengono creati eseguendo più volte lo stesso sorgente
 - fintantoché questi processi restano in esecuzione, è più efficiente che condividano il codice sorgente, visto che è lo stesso
- Ci sono anche casi in cui processi diversi vengono esplicitamente programmati per accedere a sezioni di memoria comuni
 - usando chiamate di sistema...
 - è anche il caso di thread di uno stesso processo

Requisiti: Organizzazione Logica

- A livello hardware, la memoria è organizzata in modo lineare
 - sia RAM che disco
- A livello software, i programmi sono scritti in moduli
 - i moduli possono essere scritti e compilati separatamente
 - a ciascun modulo possono essere dati diversi permessi (sola lettura, sola esecuzione)
 - i moduli possono essere condivisi tra i processi
- Il SO deve offrire tali caratteristiche, facendo da ponte tra la prima visuale e la seconda
 - spesso tramite segmentazione

Requisiti: Organizzazione Fisica

- Gestione del flusso tra RAM (piccola, veloce e volatile) e memoria secondaria (grande, lenta e permanente)
- Non può essere lasciata al programmatore:
 - la memoria potrebbe non essere sufficiente a contenere il programma ed i suoi dati
 - la tecnica dell'*overlaying* (sovrapposizione) permette a più moduli di essere posizionati nella stessa zona di memoria (in tempi diversi...), ma è difficile da programmare
 - il programmatore non sa quanta memoria avrà a disposizione
- Ci deve pensare il SO

Roadmap

- Gestione della memoria: requisiti di base
- **Partizionamento della memoria**
- Paginazione e segmentazione
- Memoria virtuale: hardware e strutture di controllo
- Memoria virtuale e sistema operativo
- Gestione della memoria in Linux

Partizionamento

- Uno dei primi metodi per la gestione della memoria
 - antecedente all'introduzione della memoria virtuale
 - non più molto usata
- Comunque utile per capire la memoria virtuale
 - la memoria virtuale è l'evoluzione moderna delle tecniche di partizionamento

Tipi di Partizionamento

- Partizionamento fisso
- Partizionamento dinamico
- Paginazione semplice
- Segmentazione semplice
- Paginazione con memoria virtuale
- Segmentazione con memoria virtuale

Partizionamento Fisso Uniforme

- Partizioni di ugual lunghezza
 - se un processo ha una dimensione minore o uguale della misura di una partizione, allora può essere caricato in una partizione libera
- Il sistema operativo può fare togliere un processo da una partizione
 - ad esempio, se nessuno dei processi attualmente in memoria è in stato *ready*

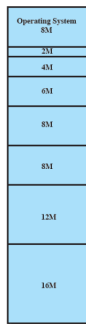


Partizionamento: Problemi

- Un programma potrebbe non entrare in una partizione
 - sta(va) al programmatore dividere il suo programma in overlays
- Uso inefficiente della memoria
 - ogni programma, anche il più piccolo, occupa un'intera partizione
 - → *frammentazione interna*

Partizioni Fisso Variabile

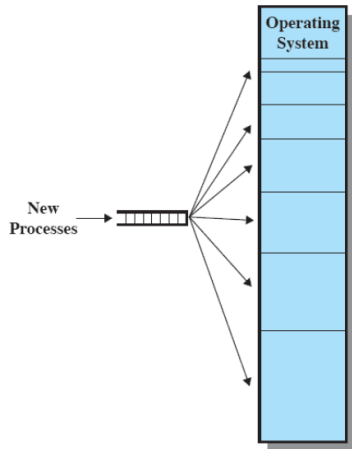
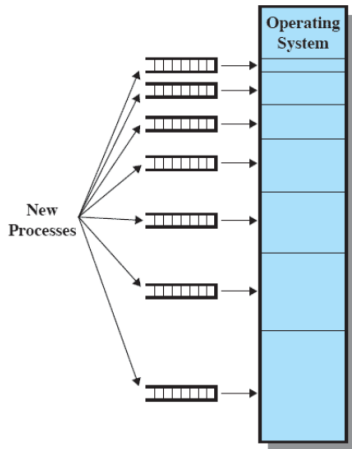
- Mitiga entrambi i problemi
 - ma non li risolve
- Nella figura, programmi più piccoli di 16M possono essere gestiti senza overlay
- Per quelli più piccoli, ci sono le partizioni più piccole
- È sempre partizionamento fisso: le partizioni sono quelle decise all'inizio e non cambiano più



Algoritmo di Posizionamento

- Partizioni di ugual lunghezza
 - algoritmo banale, non c'è scelta
- Partizioni di diversa lunghezza
 - un processo va nella partizione più piccola che può contenerlo
 - questo minimizza la quantità di spazio sprecato
 - una coda per ogni partizione, oppure una per tutte

Partizionamento Fisso e Code



Partizionamento Fisso: Problemi Irrisolti

- C'è un numero massimo di processi in memoria principale
 - corrispondente al numero di partizioni deciso inizialmente
- Se ci sono molti processi piccoli, la memoria verrà usata in modo inefficiente
 - sia con le partizioni di lunghezza uguale che con quelle variabili

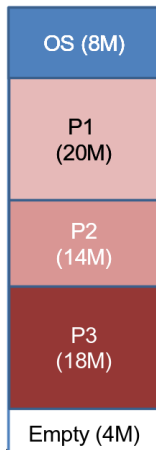
Partizionamento Dinamico

- Le partizioni variano sia in misura che in quantità
- Per ciascun processo viene allocata esattamente la quantità di memoria che serve

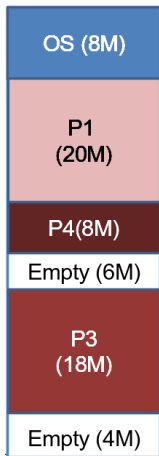
Partizionamento Dinamico: Esempio



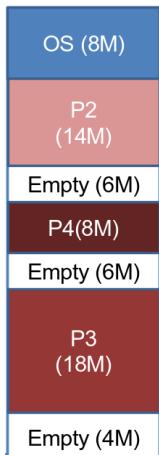
Partizionamento Dinamico: Esempio



Partizionamento Dinamico: Esempio



Partizionamento Dinamico: Esempio



Partizionamento Dinamico

- *Frammentazione esterna*: la memoria che non è usata per nessun processo viene frammentata
- Si può risolvere con la *compattazione*
 - il SO sposta i processi di modo che siano contigui
 - però ha un elevato overhead

Partizionamento Dinamico

- Il SO deve decidere a quale blocco libero assegnare un processo
- Algoritmo **best-fit** (il migliore tra quelli adatti)
 - sceglie il blocco la cui misura è la più vicina (in eccesso) a quella del processo da posizionare
 - nonostante l'apparenza ragionevole, è quello con risultati peggiori
 - lascia dei frammenti molto piccoli
 - costringe a fare spesso la compattazione

Partizionamento Dinamico

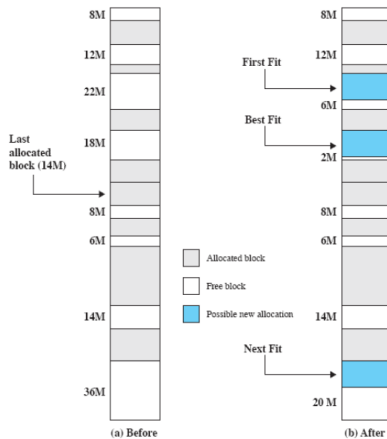
- Algoritmo **first-fit** (il primo tra quelli adatti)
 - scorre la memoria dall'inizio; il primo blocco con abbastanza memoria viene subito scelto
 - molto veloce
 - tende a riempire solo la prima parte della memoria
- A conti fatti, è(ra) il migliore

Partizionamento Dinamico

- Algoritmo **next-fit** (il prossimo tra quelli adatti)
 - come il first-fit, ma anziché partire ogni volta dall'inizio, parte dall'ultima posizione assegnata ad un processo
 - assegna più spesso il blocco alla fine della memoria, che è quello più grosso
 - il blocco più grande viene spezzato in blocchi più piccoli
 - occorre usare la compattazione per riottenere un blocco grande alla fine della memoria

Allocazione: Esempio

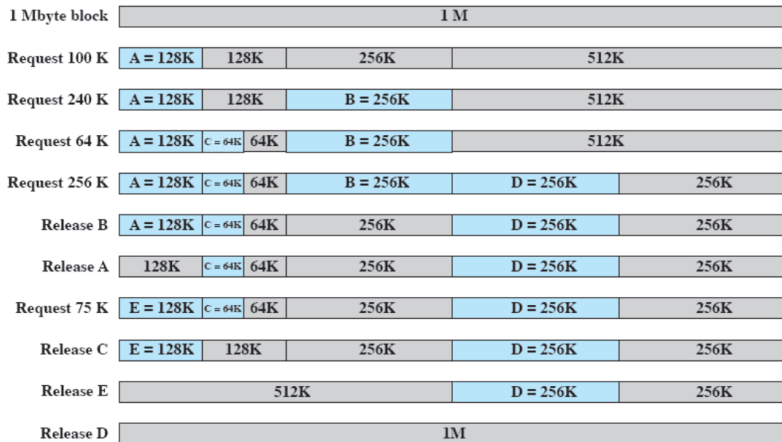
La memoria prima e dopo l'allocazione di un blocco da 16M



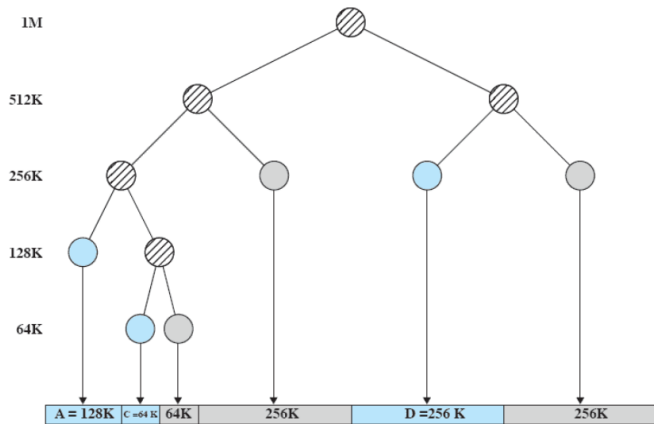
Buddy System (Sistema del Compagno)

- Compromesso tra partizionamento fisso e dinamico
- Sia 2^U la dimensione dello user space ed s la dimensione di un processo da mettere in RAM
- Si dimezza lo spazio fino a trovare un X t.c. $2^{X-1} < s \leq 2^X$, con $L \leq X \leq U$
 - una delle 2 porzioni è usata per il processo
 - L serve per dare un lower bound: non si potranno creare partizioni troppo piccole
- Ovviamente, occorre tener presente le porzioni già occupate
- Quando un processo finisce, se il buddy è libero si può fare una fusione

Esempio di Buddy System



Esempio di Buddy System: Rappresentazione ad Albero



Roadmap

- Gestione della memoria: requisiti di base
- Partizionamento della memoria
- **Paginazione e segmentazione**
- Memoria virtuale: hardware e strutture di controllo
- Memoria virtuale e sistema operativo
- Gestione della memoria in Linux

Paginazione (Semplice)

- Non usata, ma importante per introdurre la memoria virtuale
- La memoria viene partizionata in pezzi di grandezza uguale e piccola
- Lo stesso trattamento viene riservato ai processi
- I pezzi di processi (in generale, in memoria ausiliaria) sono chiamati *pagine*
- I pezzi di memoria sono chiamati *frame*
- Ogni pagina, per essere usata, dev'essere collocata in un frame
 - pagine contigue possono essere messe in frame distanti
 - in generale, una pagina può essere messa in un *qualunque* frame
 - ovviamente, una pagina ed un frame hanno la stessa dimensione

Paginazione

- I SO che la adottano mantengono una tabella delle pagine per ogni processo
- Per ogni pagina del processo, questa tabella dice in quale frame effettivo si trova
- Un indirizzo di memoria può essere visto come un numero di pagina e uno spiazzamento al suo interno
 - realizza anche la rilocazione, aggiornando lo schema con il solo base register
- Quando c'è un process switch, la tabella delle pagine del nuovo processo deve essere ricaricata

Paginazione: Esempio

Frame number	Main memory
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	

Paginazione: Esempio

Frame number	Main memory
0	A.0
1	A.1
2	A.2
3	A.3
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	

Paginazione: Esempio

Frame number	Main memory
0	A.0
1	A.1
2	A.2
3	A.3
4	B.0
5	B.1
6	B.2
7	C.0
8	C.1
9	C.2
10	C.3
11	
12	
13	
14	

Paginazione: Esempio

Frame number	Main memory
0	A.0
1	A.1
2	A.2
3	A.3
4	
5	
6	
7	C.0
8	C.1
9	C.2
10	C.3
11	
12	
13	
14	

Paginazione: Esempio

Frame number	Main memory
0	A.0
1	A.1
2	A.2
3	A.3
4	D.0
5	D.1
6	D.2
7	C.0
8	C.1
9	C.2
10	C.3
11	D.3
12	D.4
13	
14	

Paginazione: Esempio

Frame number	Main memory
0	A.0
1	A.1
2	A.2
3	A.3
4	D.0
5	D.1
6	D.2
7	C.0
8	C.1
9	C.2
10	C.3
11	D.3
12	D.4
13	
14	

Notare che, con il partizionamento dinamico, non sarebbe stato possibile caricare D in memoria

Paginazione: Esempio

Tabelle delle pagine risultanti

0	0
1	1
2	2
3	3

Process A
page table

0	—
1	—
2	—

Process B
page table

0	7
1	8
2	9
3	10

Process C
page table

0	4
1	5
2	6
3	11
4	12

Process D
page table

13
14

Free frame
list

Paginazione: Esempio di Traduzione

- Supponiamo che la dimensione di una pagina sia 100 bytes
- Quindi, la RAM dell'esempio appena visto è solo di 1400 bytes in totale
- Inoltre, i processi A, B, C, D richiedono solo 400, 300, 400 e 500 bytes rispettivamente
 - comprensivi di codice (program), dati globali ed heap (data) e stack delle chiamate
- Nelle istruzioni dei processi, i riferimenti alla RAM sono relativi all'inizio del processo
 - quindi, ad esempio, per D ci saranno solo riferimenti compresi nell'intervallo [0, 499]

Paginazione: Esempio di Traduzione

- Supponiamo ora che attualmente il processo D sia in esecuzione, e che occorra eseguire l'istruzione $j = 343$
 - vale lo stesso anche per istruzioni di load o store, anche se devono passare per registri
- Ovviamente, non si tratta dell'indirizzo 343 della RAM: lì c'è il processo A!
- Bisogna prima capire in quale pagina di D si trova 343: basta fare $343 \text{ div } 100 = 3$
- Poi occorre guardare la tabella delle pagine di D: la pagina 3 corrisponde al frame di RAM numero 11
- Il frame 11 ha indirizzi che vanno da 1100 a 1199: qual è quello giusto?
- Basta fare $343 \text{ mod } 100 = 43$, quindi il 44-esimo byte
- L'indirizzo vero è pertanto $11 \cdot 100 + 43 = 1143$

Paginazione: Esempio di Traduzione

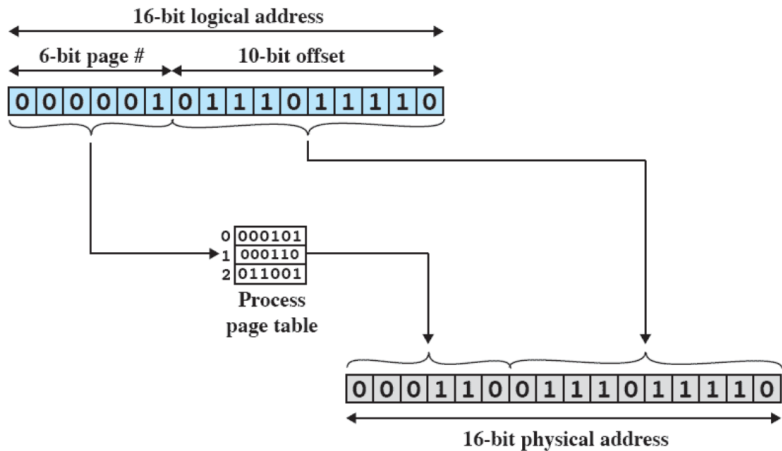
- Da notare che l'operazione di divisione per 100 equivale a “troncare” il dividendo delle 2 cifre a destra
- Mentre l'operazione di modulo lascia solo le 2 cifre a destra
- Sarebbe vero in generale con 10^k , solo che bisogna considerare k cifre anziché 2
- Quindi, visivamente:
 - 1 prendi 343 e considera solo le cifre dalla terza in poi: 3
 - 2 usa il risultato per accedere alla tabella delle pagine e prendi il frame corrispondente: 11
 - 3 sostituisci, in 343, le cifre dalla terza in poi con il numero appena trovato: 1143

Paginazione: Esempio di Traduzione

- Quanto detto sopra funziona grazie al fatto che:
 - 1 usiamo la notazione posizionale in base 10
 - 2 la dimensione delle pagine è una potenza di 10 ($100 = 10^2$)
- Per farlo funzionare anche con l'hardware dei calcolatori, bisogna usare la base 2
- Quindi la dimensione delle pagine è una potenza di 2
- E lo stesso giochetto si fa, anziché con le cifre decimali, con i bit (cifre binarie)

Paginazione

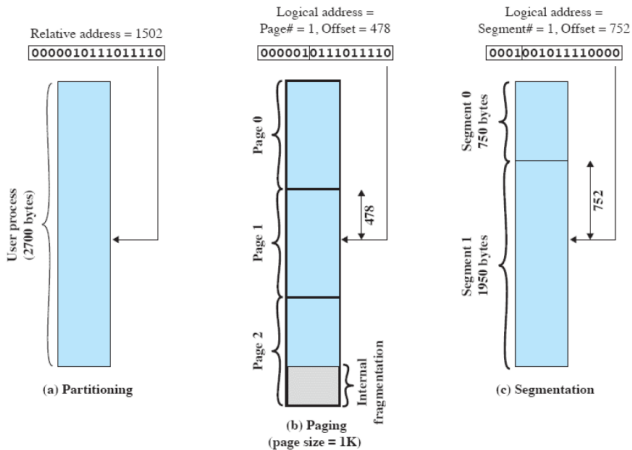
Per ogni processo, il numero di pagine è al più il numero di frames
(non sarà più vero con la memoria virtuale)



Segmentazione (Semplice)

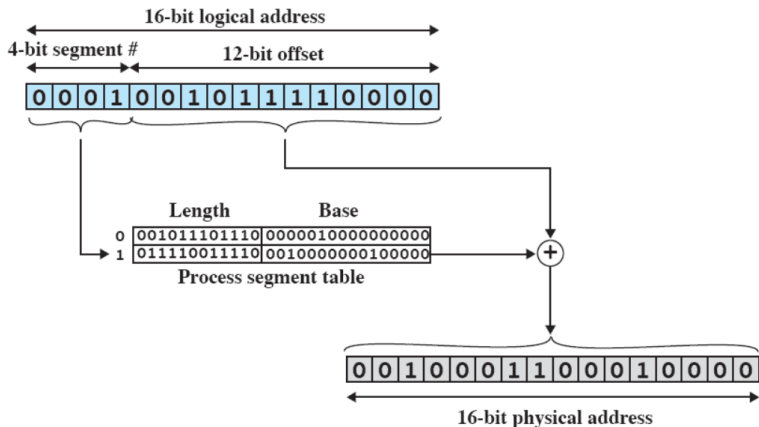
- Un programma può essere diviso in segmenti
 - i segmenti hanno una lunghezza variabile e un limite massimo alla dimensione
- Un indirizzo di memoria è un numero di segmento e uno spiazamento al suo interno
- Simile al partizionamento dinamico
 - ma con una differenza fondamentale: il programmatore (o il compilatore) devono gestire esplicitamente la segmentazione
 - dicendo quanti segmenti ci sono e qual è la loro dimensione
 - a metterli effettivamente in RAM e a risolvere gli indirizzi ci pensa il SO
 - sempre con aiuto hardware

Indirizzi Logici



Segmentazione

Qui si suppone che non possano esserci segmenti più grandi di 2^{12} bytes



(b) Segmentation

Roadmap

- Gestione della memoria: requisiti di base
- Partizionamento della memoria
- Paginazione e segmentazione
- **Memoria virtuale: hardware e strutture di controllo**
- Memoria virtuale e sistema operativo
- Gestione della memoria in Linux

Gestione della Memoria: Concetti Fondamentali

- I riferimenti alla memoria avvengono tramite indirizzi logici
 - tradotti in indirizzi fisici a tempo di esecuzione
 - questo perché un processo potrebbe essere spostato più volte dalla memoria principale alla secondaria e viceversa durante la sua esecuzione, ogni volta occupando zone di memoria diverse
- Un processo può essere spezzato in più parti, che non necessariamente occuperanno una zona contigua di memoria principale

L'Idea Geniale

- Se tutte le caratteristiche appena elencate sono vere, allora non occorre che tutte le pagine (o tutti i segmenti) di un processo siano in memoria principale, per far sì che al processo venga concesso il processore
- L'unica cosa che serve è che la prossima istruzione e i dati di cui ha bisogno siano in memoria principale
- Dalla paginazione semplice alla paginazione con memoria virtuale

Esecuzione di un Processo

- Il SO porta in memoria principale *alcuni* pezzi (pagine) del programma
- Viene chiamato **resident set** (*insieme residente*) l'insieme dei pezzi del programma che si trovano in memoria principale
- Viene generato un interrupt quando al processo serve un indirizzo che non si trova in memoria principale (*page fault*)
- È una richiesta di I/O a tutti gli effetti: il SO mette il processo in modalità blocked

Esecuzione di un Processo

- Il pezzo di processo che contiene l'indirizzo logico viene portato in memoria principale
- A tal proposito il SO effettua una richiesta di lettura su disco (I/O)
- Quindi, fintantoché questa operazione non viene completata, un altro processo va in esecuzione
- E quando l'operazione viene completata, un interrupt farà sì che il processo torni ready
 - non necessariamente in esecuzione...
- Quando verrà eseguito, occorrerà *eseguire nuovamente la stessa istruzione che aveva causato il fault*
 - questa volta, la memoria c'è

Conseguenze

- Svariati processi possono essere in memoria principale
 - non (necessariamente) per intero: solo alcune parti di ciascun processo
 - sicuramente di più che con paginazione o segmentazione semplici
- Questo vuol dire che è molto probabile che ci sia sempre almeno un processo ready
 - il processore è usato al meglio, senza diventare idle
- Un processo potrebbe anche richiedere più dell'intera memoria principale

Memoria Virtuale: Terminologia

Memoria virtuale: schema di allocazione di memoria, in cui la memoria secondaria può essere usata come se fosse principale

- gli indirizzi usati nei programmi e quelli usati dal sistema sono diversi
- c'è una fase di traduzione automatica dai primi nei secondi
- la dimensione della memoria virtuale è limitata dallo schema di indirizzamento, oltre che ovviamente dalla dimensione della memoria secondaria
- la dimensione della memoria principale, invece, non influisce sulla dimensione della memoria virtuale

Memoria Virtuale: Terminologia

Indirizzo virtuale: l'indirizzo associato ad una locazione della memoria virtuale

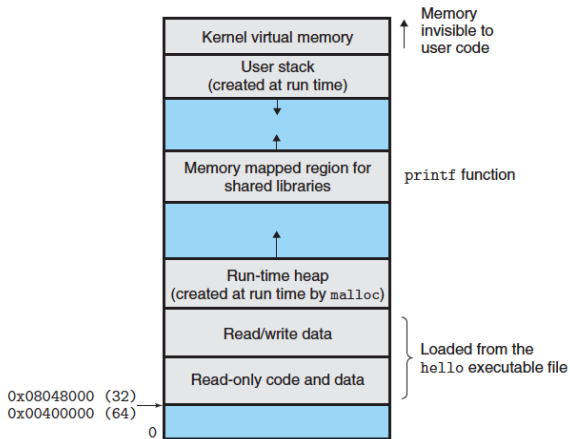
- fa sì che si possa accedere a tale locazione come se fosse parte della memoria principale

Spazio degli indirizzi virtuali: la quantità di memoria virtuale assegnata ad un processo

Spazio degli indirizzi: la quantità di memoria assegnata ad un processo

Indirizzo reale: indirizzo di una locazione di memoria principale

Linux: Come un Processo Vede la Memoria



Memoria Reale e Virtuale

- Memoria reale: quella principale (la RAM)
- Memoria virtuale: quella secondaria (su disco)
 - permette di avere una multiprogrammazione elevata
 - libera il programmatore dai vincoli della memoria (principale)

Thrashing

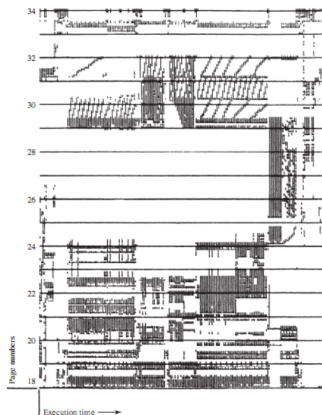
- Letteralmente: bastonatura o sconfitta
- Il SO impiega la maggior parte del suo tempo a swappare pezzi di processi, anziché ad eseguire istruzioni
- Altrimenti detto, quasi ogni richiesta di pagina dà luogo ad un page fault
- Per evitarlo, il SO cerca di indovinare quali pezzi di processo saranno usati con minore o maggiore probabilità nel futuro prossimo
 - ovvero, nella prossima istruzione da eseguire...
- Questo tentativo di divinazione avviene sulla base della storia recente

Principio di Località

- I riferimenti che un processo fa tendono ad essere vicini
 - sia che si tratti di dati che di istruzioni
- Quindi solo pochi pezzi di processo saranno necessari di volta in volta
- Quindi si può prevedere abbastanza bene quali pezzi di processo saranno necessari nel prossimo futuro
- Concludendo, la memoria virtuale può funzionare (e funziona) bene

Pagine e Località: Esempio

Di volta in volta, i riferimenti sono confinati ad un sottoinsieme delle pagine



Memoria Virtuale: Supporto Richiesto

- Paginazione e segmentazione devono essere supportati dall'hardware
 - alcune operazioni sarebbero troppo lunghe se fatte in software dal SO
 - in particolare, la traduzione degli indirizzi è hardware
- Il SO deve essere in grado di muovere pagine e/o segmenti dalla memoria principale alla secondaria

Paginazione

- Ogni processo ha una sua tabella delle pagine
 - il control block di un processo punta a tale tabella
- Ogni entry di questa tabella contiene:
 - il numero di frame in memoria principale
 - non c'è il numero di pagina: è direttamente usato per indicizzare la tabella
 - un bit per indicare se è in memoria principale o no
 - un altro bit per indicare se la pagina è stata modificata in seguito all'ultima volta che è stata caricata in memoria principale

Virtual Address

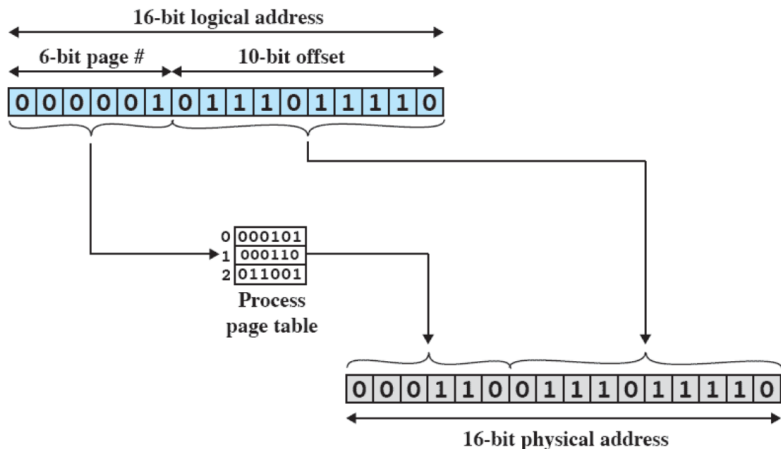
Page Number	Offset
-------------	--------

Page Table Entry

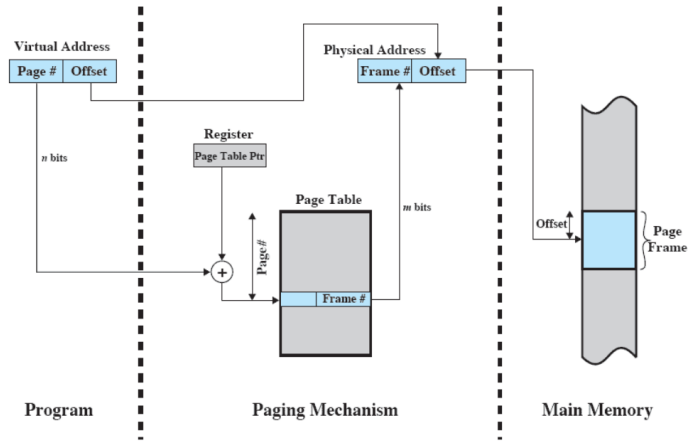
P	M	Other Control Bits	Frame Number
---	---	--------------------	--------------

Traduzione degli Indirizzi

Tipicamente, più pagine che frames, quindi non realistico lo stesso numero di bit



Traduzione degli Indirizzi



Traduzione degli Indirizzi, Caveat

- La traduzione è fatta dall'hardware
 - la somma non è una semplice somma: il numero di pagina va anche moltiplicato per il numero di bytes di ogni singola entry della tabella delle pagine
 - dopodiché , si può fare la somma
- Affinché lo schema funzioni, il sistema operativo deve:
 - caricare a partire da un certo indirizzo / la tabella delle pagine del processo
 - caricare il valore di / in un opportuno registro dipendente dall'hardware
 - CR3 nei Pentium
 - questo va fatto ad ogni process switch, quindi fa sempre parte del passo 6

Tabelle delle Pagine

- Le tabelle delle pagine potrebbero contenere molti elementi
- Quando un processo è in esecuzione, viene assicurato che almeno una parte della sua tabella delle pagine sia in memoria principale
- Qualche numero: 8GB di spazio virtuale, 1kB per ogni pagina
→ $\frac{2^{33}}{2^{10}} = 2^{23}$ entries per ogni tabella delle pagine
 - ovvero, per ogni processo
 - quanto occupa una entry? 1 byte di controllo + log(size RAM in frames)
 - con max 4GB di RAM (architettura a 32-bit) fanno 4 bytes
 - max 32 bit - 10 bit = 22 bit per i frame quindi 3 bytes, più il byte di controllo
 - fanno $4 \cdot 2^{23} = 2^{23+2} = 32\text{MB}$ di overhead per ogni processo
 - con RAM di 1 GB, bastano 20 processi per occupare più di metà RAM con sole strutture di overhead

Tabella delle Pagine a 2 Livelli

Ovviamente, il processore deve avere hardware dedicato per i 2 livelli di traduzione (il SO si deve adattare all'hardware)

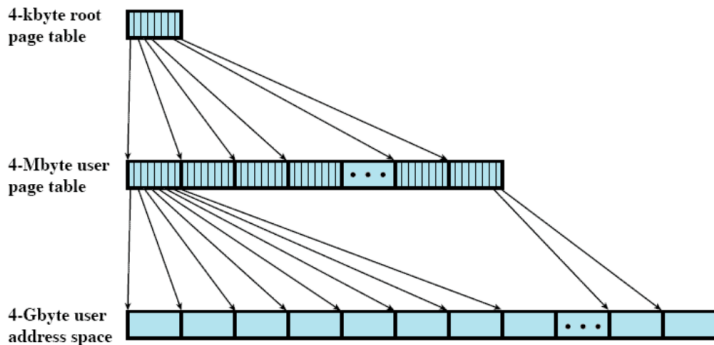


Tabella ad un Livello: Traduzione

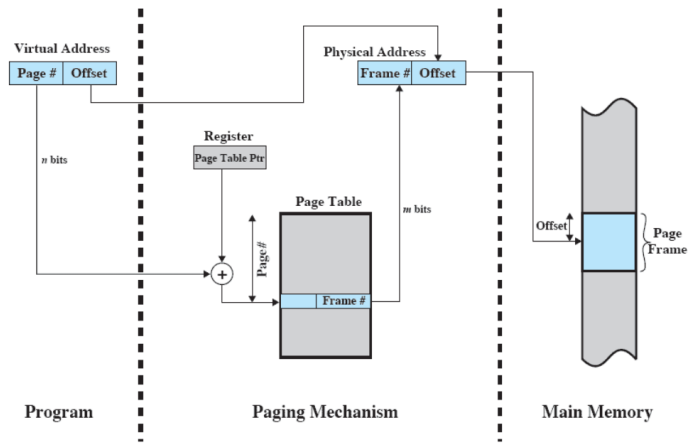


Tabella a 2 Livelli: Traduzione

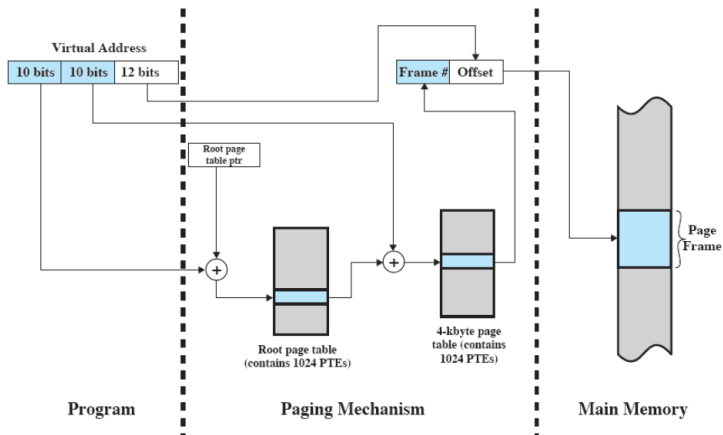


Tabelle delle Pagine a 2 Livelli

- 8GB di spazio virtuale → 33 bits di indirizzo
 - facciamo ad es. 15 bit primo livello (*directory*), 8 bit di secondo livello, e i rimanenti 10 per l'offset
 - spesso i processori impongono che una page table di secondo livello entri in una pagina (ad es. Pentium)
 - così, effettivamente, essa occupa $2^8 \cdot 2^2 = 2^{10}$ bytes
 - per ogni processo, l'overhead è $2^{23+2} = 32\text{MB}$, più l'occupazione del primo livello: $2^{15+2} = 128\text{kB}$: sempre all'incirca 32MB
 - però è più facile paginare la tabella delle pagine: in RAM basta che ci sia il primo livello più una tabella del secondo
 - quindi l'overhead scende a $2^{15+2} + 2^{8+2} = 128\text{kB}$
 - con RAM di 1 GB, occorrono 1000 processi per occupare più di metà RAM con sole strutture di overhead
 - sul mio Linux, in questo momento ci sono 338 processi in esecuzione (contando anche quelli non ready) su 8GB di RAM: l'overhead sarebbe del 5 per mille

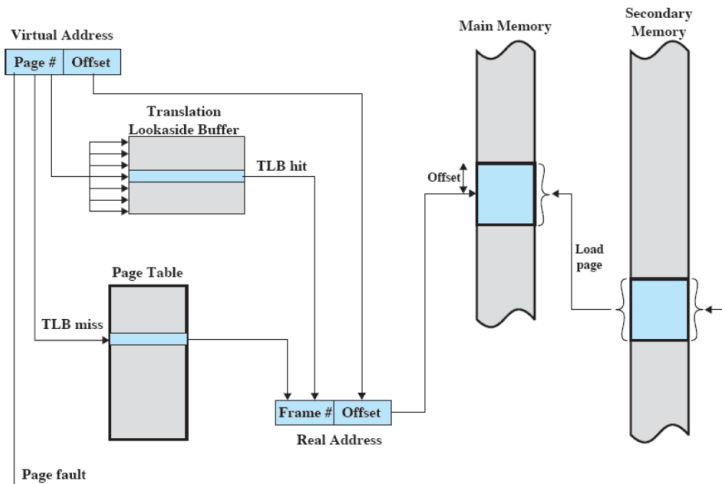
Translation Lookaside Buffer

- TLB; letteralmente: memoria temporanea per la traduzione futura
- Ogni riferimento alla memoria virtuale può generare due accessi alla memoria
 - uno per la tabella delle pagine
 - uno per prendere il dato
- Si usa una cache veloce per gli elementi delle tabelle delle pagine
 - è proprio il TLB
 - contiene gli elementi delle tabelle delle pagine che sono stati usati più di recente

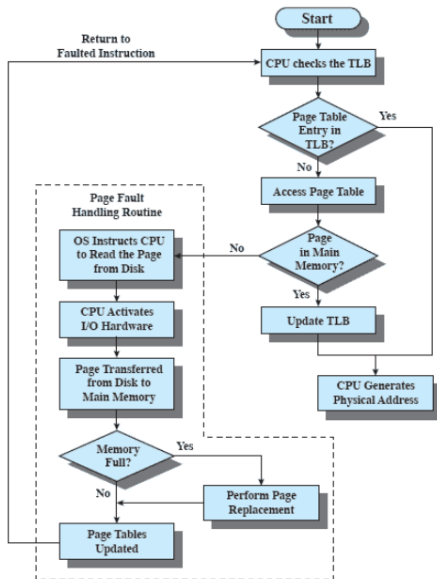
TLB: Come Funziona

- Dato un indirizzo virtuale, il processore esamina dapprima il TLB
- Se la pagina è presente (*TLB hit*), si prende il frame number e si ricava l'indirizzo reale
- Altrimenti (*TLB miss*), si prende la “normale” tabella delle pagine del processo
- Se la pagina risulta in memoria principale a posto, altrimenti si gestisce il page fault come descritto sopra
- Dopodiché, il TLB viene aggiornato includendo la pagina appena acceduta
 - usando un qualche algoritmo di rimpiazzamento se il TLB è già pieno: solitamente LRU

TLB: Come Funziona



TLB: Come Funziona



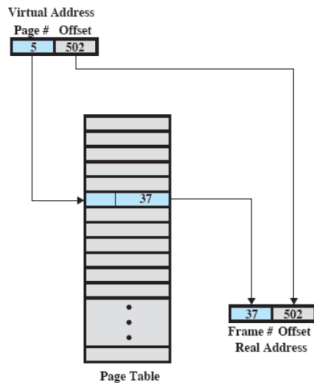
Memoria Virtuale e Process Switch

- Il sistema operativo deve poter resettare il TLB
 - è la soluzione peggiore dal punto di vista delle prestazioni
- Per far almeno un po' meglio, alcuni processori permettono:
 - di etichettare con il PID ciascuna entry del TLB (es. Pentium)
 - di invalidare solo alcune parti del TLB (raro, alla fine inefficiente)
- È comunque necessario, anche senza TLB, dire al processore dove è la nuova tabella delle pagine
 - nel caso sia a 2 livelli, basta la page directory
 - gli indirizzi vanno caricati in opportuni registri
 - fa parte del process switch: aggiorna quanto detto per la rilocalizzazione semplice

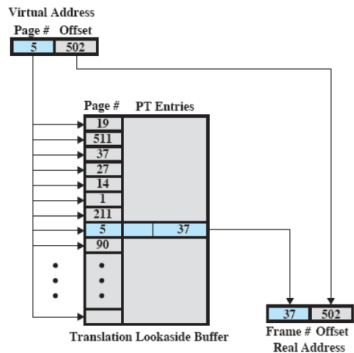
Mapping Associativo

- Il TLB contiene solo *alcuni* elementi tratti dalle tabelle delle pagine, il numero della pagina non può essere usato direttamente come indice per il TLB
 - cosa invece possibile nella tabella delle pagine, che le ha tutte
- Il SO può interrogare più elementi del TLB contemporaneamente per capire se c'è o no un TLB hit
 - c'è supporto hardware per fare ciò
- Altro problema: bisogna fare in modo che il TLB contenga solo pagine *in RAM*
 - altrimenti, page fault dopo un TLB hit, ma sarebbe impossibile accorgersene
 - non si guarda la tabella delle pagine!
 - tornano qui utili le istruzioni hardware di reset parziale del TLB
 - se il SO operativo swappa una pagina, deve anche istruire il TLB di eliminarla

TLB e Mapping Associativo

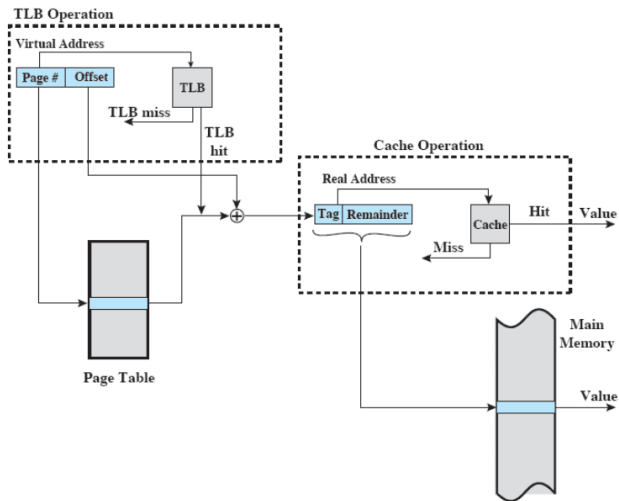


(a) Direct mapping



(b) Associative mapping

TLB e Cache



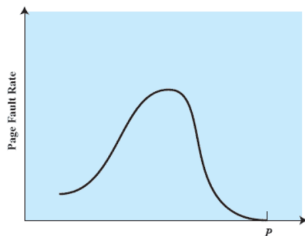
Dimensione delle Pagine

- Più piccola è una pagina, minore è la frammentazione all'interno delle pagine
- Ma è anche maggiore il numero di pagine per processo
- Il che significa che è più grande la tabella delle pagine (per ogni processo)
- E quindi la maggior parte delle tabelle delle pagine finisce in memoria secondaria
- La memoria secondaria è ottimizzata per trasferire grossi blocchi di dati, quindi avere le pagine ragionevolmente grandi non sarebbe male

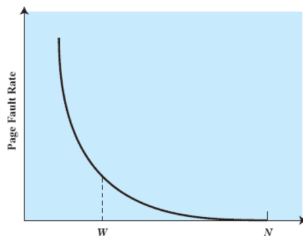
Dimensione delle Pagine

- Più piccola è una pagina, maggiore il numero di pagine che si trovano in memoria principale
- E in tutte queste pagine, i riferimenti saranno vicini: in accordo con la località, i page fault saranno pochi
- Con pagine più grandi, tutto questo cade, e i page fault saranno maggiori

Page Faults vs. Dimensione Pagina



(a) Page Size



(b) Number of Page Frames Allocated

P = size of entire process
 W = working set size
 N = total number of pages in process

Con pagine grandi, pochi fault di pagina, ma poca multiprogrammazione!

Dimensione delle Pagine in Alcuni Sistemi

Computer	Page Size
Atlas	512 48-bit words
Honeywell-Multics	1024 36-bit word
IBM 370/XA and 370/ESA	4 Kbytes
VAX family	512 bytes
IBMAS/400	512 bytes
DEC Alpha	8 Kbytes
MIPS	4 Kbytes to 16 Mbytes
UltraSPARC	8 Kbytes to 4 Mbytes
Pentium	4 Kbytes or 4 Mbytes
IBMPower	4 Kbytes
Itanium	4 Kbytes to 256 Mbytes

Dimensione delle Pagine in Alcuni Sistemi

- Le moderne architetture HW possono supportare diverse dimensioni delle pagine (anche fino ad 1GB)
- Il sistema operativo ne sceglie una: Linux sugli x86 va con 4kB
- Le dimensioni più grandi sono usate in sistemi operativi di architetture grandi: cluster, grandi server, ma anche per i sistemi operativi stessi (kernel mode)

Computer	Page Size
Atlas	512 48-bit words
Honeywell-Multics	1024 36-bit word
IBM 370/XA and 370/ESA	4 Kbytes
VAX family	512 bytes
IBMAS/400	512 bytes
DEC Alpha	8 Kbytes
MIPS	4 Kbytes to 16 Mbytes
UltraSPARC	8 Kbytes to 4 Mbytes
Pentium	4 Kbytes or 4 Mbytes
IBMPowerPC	4 Kbytes
Itanium	4 Kbytes to 256 Mbytes

Roadmap

- Gestione della memoria: requisiti di base
- Partizionamento della memoria
- Paginazione e segmentazione
- Memoria virtuale: hardware e strutture di controllo
- Memoria virtuale e sistema operativo
- Gestione della memoria in Linux

Segmentazione

- Permette al *programmatore* di vedere la memoria come un insieme di spazi (segmenti) di indirizzi
- La dimensione degli indirizzi può essere variabile ed anche dinamica
- Semplifica la gestione delle strutture dati che crescono
- Permette di modificare e ricompilare i programmi in modo indipendente
- Permette di condividere dati
- Permette di proteggere dati

Segmentazione: Organizzazione

- Ogni processo ha una sua tabella dei segmenti
 - il control block di un processo punta a tale tabella
- Ogni entry di questa tabella contiene:
 - l'indirizzo di partenza (in memoria principale) del segmento
 - la lunghezza del segmento
 - un bit per indicare se il segmento è in memoria principale o no
 - un altro bit per indicare se il segmento è stato modificato in seguito all'ultima volta che è stato caricato in memoria principale

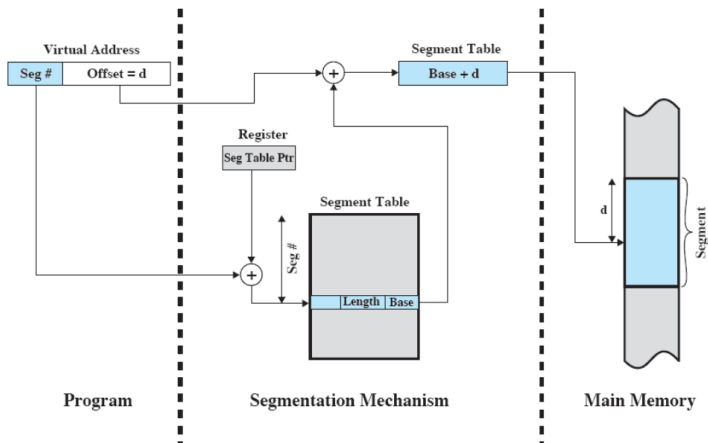
Virtual Address

Segment Number	Offset
----------------	--------

Segment Table Entry

Present	Other Control Bits	Length	Segment Base
---------	--------------------	--------	--------------

Segmentazione: Traduzione degli Indirizzi



Paginazione e Segmentazione

- La paginazione è trasparente al programmatore
 - nel senso che il programmatore non ne è (o non ne deve essere) a conoscenza
 - vale anche per il compilatore
- La segmentazione è visibile al programmatore
 - ovviamente, se programma in assembler
 - altrimenti, ci pensa il compilatore ad usare i segmenti
- Ogni segmento viene diviso in più pagine

Paginazione e Segmentazione

Virtual Address



Segment Table Entry

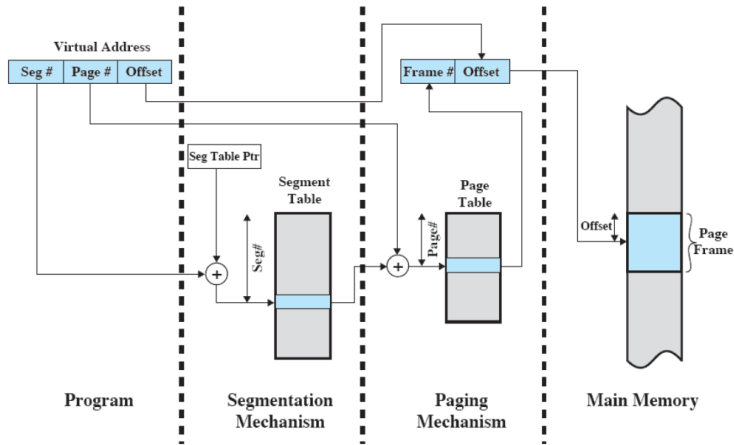


Page Table Entry



P= present bit
M = Modified bit

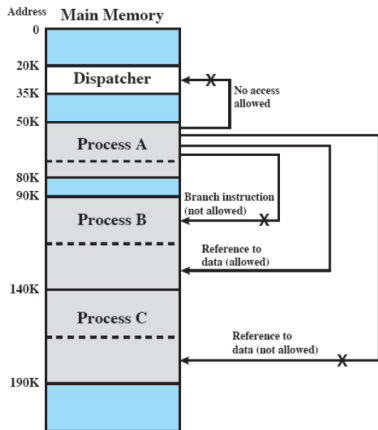
Paginazione e Segmentazione: Traduzione degli Indirizzi



Protezione e Condivisione

- Con la segmentazione, implementare protezione e condivisione viene naturale
- Dato che ogni segmento ha una base ed una lunghezza, è facile controllare che i riferimenti siano contenuti nel giusto intervallo
- Per la condivisione, basta dire che uno stesso segmento serve più processi

Protezione



Roadmap

- Gestione della memoria: requisiti di base
- Partizionamento della memoria
- Paginazione e segmentazione
- Memoria virtuale: hardware e strutture di controllo
- **Memoria virtuale e sistema operativo**
- Gestione della memoria in Linux

Gestione della Memoria: Decisioni

- Usare o no la memoria virtuale?
- Usare solo la paginazione?
- Usare solo la segmentazione?
- Usare paginazione e segmentazione?
- Che algoritmi usare per gestire i vari aspetti della gestione della memoria?

Elementi Centrali per il Progetto del SO

- Politica di prelievo (*fetch policy*)
- Politica di posizionamento (*placement policy*)
- Politica di sostituzione (*replacement policy*)
- Altro (solo cenni):
 - gestione del resident set
 - politica di pulitura
 - controllo del carico
- Il tutto, cercando di minimizzare i page fault; non c'è una politica sempre vincente

- Decide quando una pagina data debba essere portata in memoria principale
- Si usano principalmente due politiche:
 - paginazione su richiesta (*demand paging*)
 - prepaginazione (*prepaging*)

Demand Paging e Prepaging

- Demand paging:
 - una pagina viene portata in memoria principale nel momento in cui un qualche processo la richiede
 - molti page fault nei primi momenti di vita del processo
- Prepaging:
 - porta in memoria principale più pagine di quelle richieste
 - ovviamente, si tratta di pagine vicine a quella richiesta (si può fare efficientemente sul disco)

Dove si Mette una Pagina?

- Placement policy: decide dove mettere una pagina in memoria principale *quando c'è almeno un frame libero*
 - se non ci sono frame liberi, allora *replacement policy*
- C'è l'hardware per la traduzione degli indirizzi, quindi può essere messa ovunque
- Tipicamente, il primo frame libero è quello dove viene messa la pagina
 - “primo” → con indirizzo più basso

Quale Pagina Sostituire?

- Cosa fare se, quando occorre portare una pagina in memoria principale, tutti i frame sono già occupati?
- Come con la cache: qualche pagina va sostituita, sovrascrivendo il rispettivo frame
- Replacement policy: decide quale pagina sostituire
- Va fatto in modo da minimizzare la probabilità che la pagina appena sostituita venga subito richiesta di nuovo
- Usando il principio di località, si cerca di predire il futuro sulla base del passato recente

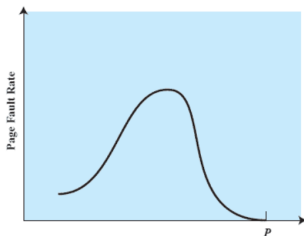
Gestione del Resident Set

- Risponde a 2 necessità:
 - per ogni processo in esecuzione (*attivo*), quanti frame di RAM vanno allocati?
 - *resident set management* propriamente detto
 - quando si rimpiazza un frame, bisogna scegliere solo tra i frame che appartengono al processo corrente, oppure si può sostituire un frame qualsiasi?
 - *replacement scope*

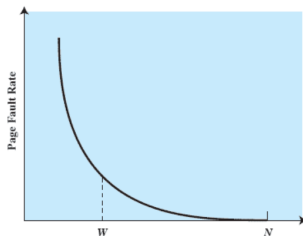
Gestione del Resident Set

- Ci sono 2 tecniche per ogni problema
- Dimensione del resident set:
 - allocazione fissa
 - il numero di frame è deciso al tempo di creazione di un processo
 - allocazione dinamica
 - il numero di frame varia durante la vita del processo
 - magari basandosi sulle statistiche che man mano vengono raccolte

Page Faults vs. Dimensione Pagine



(a) Page Size



(b) Number of Page Frames Allocated

P = size of entire process
 W = working set size
 N = total number of pages in process

Ovviamente, con resident set alto ottimi page fault rate
Ma di nuovo poca multiprogrammazione...

Gestione del Resident Set

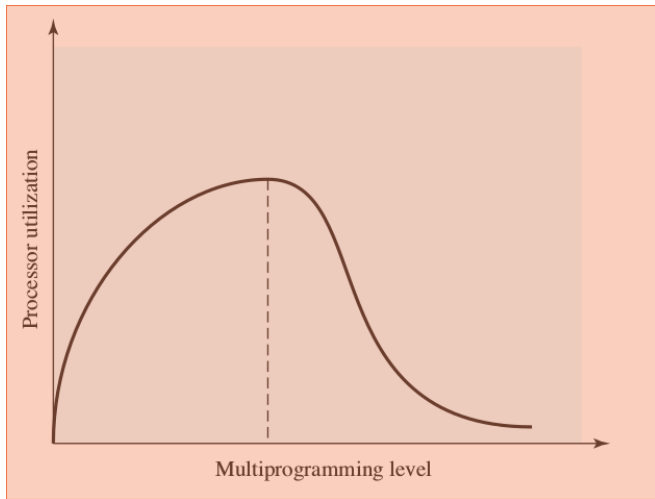
- Replacement Scope:
 - politica locale
 - se bisogna rimpiazzare un frame, si sceglie un altro frame dello stesso processo
 - politica globale
 - si può scegliere qualsiasi frame (non del SO...)
- In tutto fanno... 3 possibili strategie
 - con l'allocazione fissa, la politica globale non si può usare
 - altrimenti, si potrebbe ampliare il numero di frames di un processo, e non sarebbe più allocazione fissa

Replacement Policy: Frame Bloccati

- **Frame Locking**: se un frame è bloccato, non si può sostituire
- Si fa a livello di kernel del sistema operativo
- È sufficiente assegnare un bit ad ogni frame
- Vengono bloccati i frame del sistema operativo, ed eventualmente quelli di altri processi
 - se si usa la politica locale per il rimpiazzamento

- Se un frame è stato modificato, va riportata la modifica anche sulla pagina corrispondente
- Il problema è: quando?
 - non appena avviene la modifica
 - non appena il frame viene sostituito
- Si fa tipicamente una via di mezzo, intrecciata con il page buffering (vedere più avanti)
 - solitamente, si raccolgono un po' di richieste di frame da modificare e li si esegue

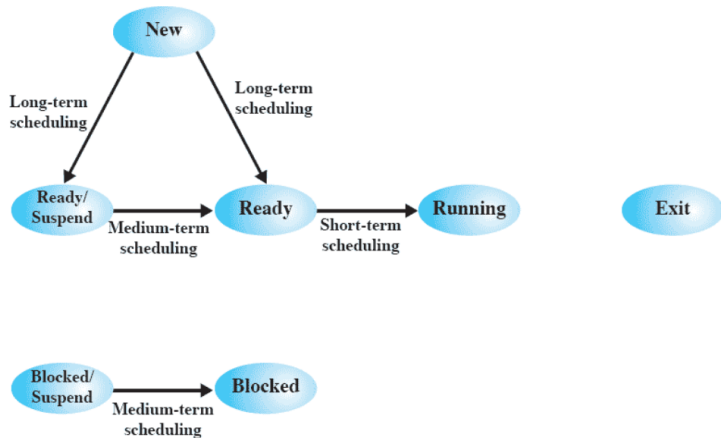
Controllo del Carico (Medium-Term Scheduler)



Multiprogrammazione da tenere non troppo alta: il resident set di ogni processo diverrebbe troppo basso → troppi page fault

Stati dei Processi e Scheduling

Il medium-term scheduler può anche sospendere processi (qui non messo perché non è l'unico motivo)



Controllo del Carico (Medium-Term Scheduler)

- Si cerca di aumentare e diminuire il numero di processi attivi
 - aumentando la multiprogrammazione, ma senza arrivare al thrashing
 - si aumenta svegliando dalla sospensione, si diminuisce sospendendo
- Politiche di monitoraggio
 - ad es.: si aggiusta la multiprogrammazione in modo che il tempo medio tra 2 fault è uguale al tempo medio di gestione di un fault
- Invocato ogni tot page fault, fa parte dell'algoritmo di rimpiazzamento

Controllo del Carico (Medium-Term Scheduler)

- Come si sceglie un processo da sospendere?
 - processo con minore priorità
 - processo che ha causato l'ultimo page fault
 - ultimo processo attivato
 - processo con il working set più piccolo
 - processo con immagine più grande
 - processo con il più alto tempo rimanente di esecuzione (se disponibile)

Algoritmi di Sostituzione

- Sostituzione ottima
- Sostituzione della pagina usata meno di recente (*LRU: Least Recently Used*)
- Sostituzione a coda (*FIFO: First In First Out*)
- Sostituzione ad orologio (*clock*)

Algoritmi di Sostituzione

- Gli esempi riportati nel seguito usano tutti la stessa sequenza di richieste a pagine:

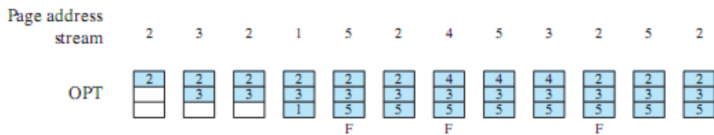
2 3 2 1 5 2 4 5 3 2 5 2

- Si suppone inoltre che ci siano solo 3 frame in memoria principale

Sostituzione Ottimale

- Si sostituisce la pagina che verrà richiesta più in là nel futuro
- Ovviamente, non è implementabile
- È però definibile sperimentalmente
- Usata per confronti sperimentali

Sostituzione Ottimale sull'Esempio



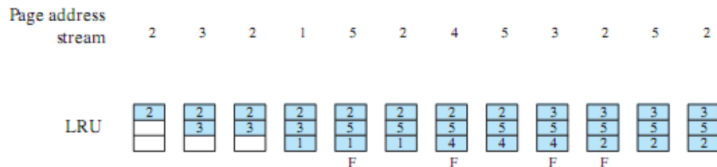
F = page fault occurring after the frame allocation is initially filled

Risultato: 3 page faults

Sostituzione LRU

- Sostituisce la pagina cui non sia stato fatto riferimento per il tempo più lungo
- Basandosi sul principio di località, dovrebbe essere la pagina che ha meno probabilità di essere usata nel prossimo futuro
- Implementazione problematica
 - occorre etichettare ogni frame con il tempo dell'ultimo accesso...
 - la cache usa questa tecnica perché è implementata in hardware
 - ma non si può fare pure questo in hardware, troppo costoso

Sostituzione LRU sull'Esempio



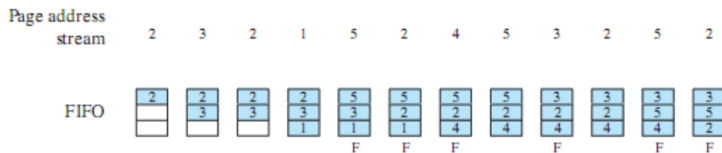
F= page fault occurring after the frame allocation is initially filled

Risultato: 4 page faults, quasi come l'ottimo

Sostituzione FIFO

- I frame allocati ad un qualche processo sono trattati come una coda circolare
- Da questa coda, le pagine vengono rimosse a turno (*round robin*)
- Implementazione semplice
- Si rimpiazzano le pagine che sono state in memoria per più tempo
 - però potrebbero servire: magari alcune di loro hanno molti accessi

Sostituzione FIFO sull'Esempio



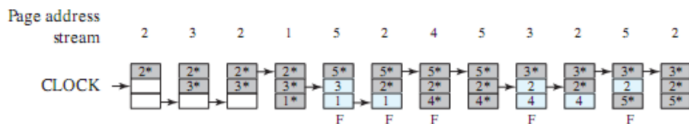
F= page fault occurring after the frame allocation is initially filled

Risultato: 6 page faults
non si accorge che la 2 e la 5 sono molto richieste

Sostituzione dell'Orologio

- Compromesso tra LRU e FIFO
- C'è uno “use bit” per ogni frame, che indica se la pagina caricata nel frame è stata riferita
- Il bit è settato ad 1 quando la pagina viene caricata in memoria principale, e poi rimesso ad 1 per ogni accesso al suo interno
- Quando occorre sostituire una pagina, il SO cerca come nella FIFO
- Ma seleziona il frame contenente la pagina che ha per prima lo use bit a 0
- Se invece incontra una pagina che lo ha a 1, lo mette a 0 e procede con la prossima

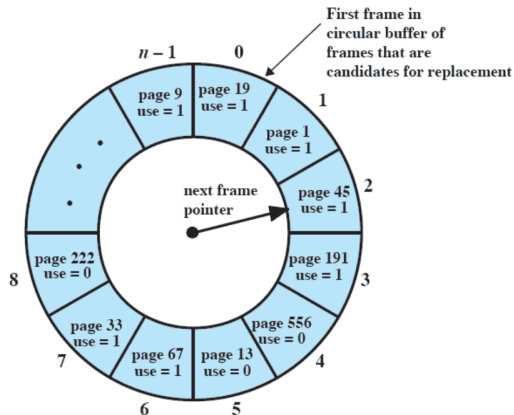
Sostituzione dell'Orologio sull'Esempio



F = page fault occurring after the frame allocation is initially filled

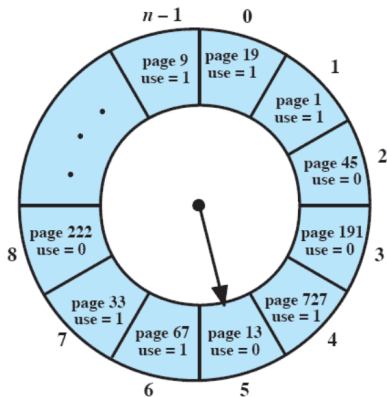
Risultato: 5 page faults
si accorge che la 2 e la 5 sono molto richieste

Politica dell'Orologio



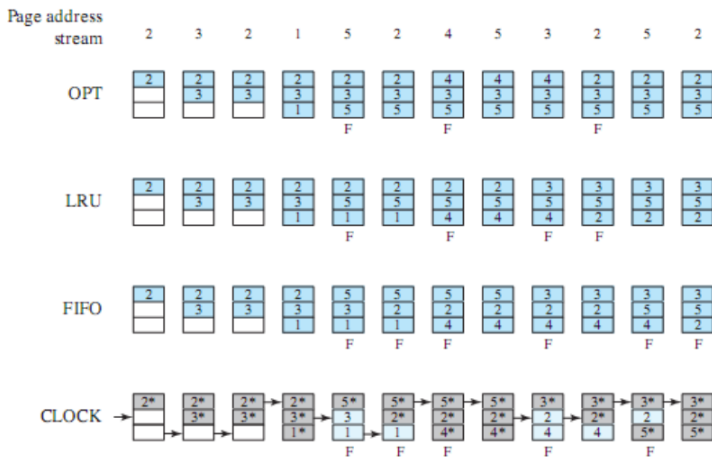
(a) State of buffer just prior to a page replacement

Politica dell'Orologio



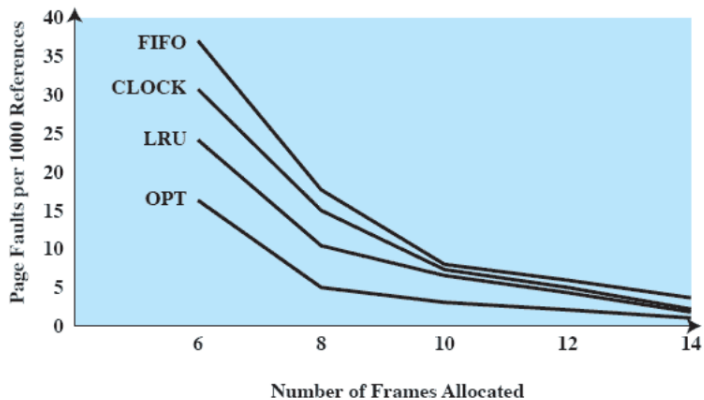
(b) State of buffer just after the next page replacement

Algoritmi di Sostituzione sull'Esempio



F = page fault occurring after the frame allocation is initially filled

Algoritmi di Sostituzione: Confronto

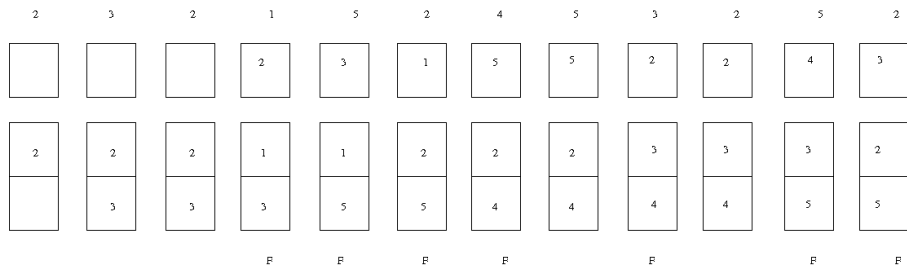


Buffering delle Pagine

- Ennesima cache (ma non hardware), stavolta per le pagine
- È una modifica del FIFO
 - ma talvolta usata anche con LRU e/o clock
 - avvicina il semplice FIFO al clock (semplice) come prestazioni
- Se occorre rimpiazzare un pagina, non viene subito buttata via, ma viene messa in questa cache
- Così se poi viene nuovamente referenziata, si può subito riportarla in memoria
- Tipicamente divisa tra pagine modificate e non
- Si cerca di scrivere le pagine modificate tutte insieme
 - anche con LRU e/o clock
 - si scrive su disco quando la lista delle pagine modificate diventa piena o quasi

FIFO e Page Buffering sull'Esempio

Assumendo che tutti gli accessi siano in lettura, e che la cache sia di 1 sola pagina



Roadmap

- Gestione della memoria: requisiti di base
- Partizionamento della memoria
- Paginazione e segmentazione
- Memoria virtuale: hardware e strutture di controllo
- Memoria virtuale e sistema operativo
- **Gestione della memoria in Linux**

Gestione della Memoria in Linux

- Distinzione netta tra richieste di memoria da parte del kernel e di processi utente
- Il kernel si fida di se stesso
- Pochi controlli, se non nessuno, per richieste da parte di se stesso
- Per i processi utente controlli di protezione e di rispetto dei limiti assegnati
 - SIGSEGV: segmentation fault...

Gestione della Memoria Kernel (Cenni)

- Ma soprattutto, è completamente diversa la gestione della memoria
- Semplificando, le richieste di memoria del kernel sono ottimizzate sia per le richieste piccole che per le grandi
- Il kernel può usare sia la memoria a lui riservata nella parte alta della memoria, che quella normalmente usata dai processi utente
 - se la richiesta è piccola (pochi bytes), fa in modo di avere alcune pagine già pronte da cui può prendere i pochi bytes richiesti (*slab allocator*)
 - se la richiesta è grande (fino a 4MB), fa in modo di allocare più pagine contigue in frame contigui
 - importante per il DMA, che ignora il fatto che ci sia la paginazione e va dritto per dritto in RAM
 - quando il kernel assegna una memoria ad un DMA, questa dev'essere quindi contigua
 - a tal proposito, usa essenzialmente il *buddy system*

Gestione della Memoria Utente

- Fetch policy: paging on demand
- Placement policy: il primo frame libero
- Replacement policy: nel seguito
- Gestione del resident set: politica dinamica con replacement scope globale
 - *molto* globale: vi rientrano anche la cache del disco (*page cache*, ci torneremo)
 - anche qui, c'è un buddy system per richieste grandi
- Politica di pulitura: ritardata il più possibile
 - scrittura quando la page cache è troppo piena con molte richieste pending
 - troppe pagine sporche, o pagina sporca da troppo tempo
 - richiesta esplicita di un processo: *flush*, *sync* o simili
- Controllo del carico: assente

Linux: Replacement Policy

- Algoritmo dell'orologio "corretto", ma il kernel più recente usa un LRU "corretto"
 - il precedente algoritmo dell'orologio era molto efficace, ma poco efficiente
 - soprattutto con le più recenti memorie da decine di GB
- Due flag in ogni entry delle page table: PG_referenced e PG_active
 - sulla base di PG_active, due liste di pagine sono mantenute dal kernel: attive ed inattive
- kswapd: kernel thread in esecuzione periodica, scorre solo le pagine inattive
 - PG_referenced è settato quando la pagina viene richiesta
 - dopodiché, delle due l'una: o arriva prima kswapd o un altro riferimento
 - nel secondo caso, pagina attiva
 - nel primo, PG_referenced di nuovo a zero
- Solo le pagine inattive possono essere rimpiazzate
 - tra di esse, si fa una mezza specie di LRU con contatore a 8 bit

Linux: Replacement Policy

