10 - Gestione Memoria

Multiprogrammazione e gestione memoria

Obiettivo primario della multiprogrammazione e' l'uso efficiente delle risorse computazionali:

- efficienza nell'uso della CPU
- velocita' di risposta dei processi
- ...

Necessita di mantenere piu' processi in memoria centrale: SO deve gestire la memoria in modo da consentire la presenza contemporanea di piu' processi

Caratteristiche importanti :

- Velocita'
- Grado di multiprogrammazione
- Utilizzo della memoria
- Protezione

Gestione della memoria centrale

A livello hw: ogni sistema e' equipaggiato con un unico spazio di memoria accessibile direttamente da CPU e dispositivi

Compiti di SO

- allocare memoria ai processi
- deallocare memoria
- separare gli spazi di indirizzi associati ai processi (protezione)
- realizzare i collegamenti (binding) tra gli indirizzi logici specificati dai processi e le corrispondenti locazioni nella memoria fisica
- memoria virtuale: gestire gli spazi di indirizzi logici di dimensioni superiori allo spazio fisico

Accesso alla memoria

Memoria centrale:

- vettore di celle, ognuna univocamente indivuduata da un indirizzo

- operazioni fondamentali sulla memoria: load/store dati e istruzioni
- Indirizzi
  - **simbolici**: riferimenti a celle di memoria nei programmi in forma sorgente mediante nomi simbolici
  - **logici**: riferimenti a celle nello spazio logico di indirizzamento del processo
  - **fisici:** riferimenti assoluti delle celle in memoria a livello HW

Indirizzi simbolici, logici e fisici

Ogni processo dispone di un proprio spazio di indirizzamento logico [0, max] che viene allocato nella memoria fisica.

Binding degli indirizzi

Ad ogni indirizzo logico/simbolico viene fatto corrispondere un indirizzo fisico: l'associzione tra indirizzi relativi e indirizzi assoluti viene detta **binding**

Binding puo' essere effettuato:

- **staticamente**
  - a tempo di compilazione: il compilatore degli indirizzi assoluti (esempio: file .com DOS)
  - a tempo di caricamento: il compilatore genera degli indirizzi relativi che vengono convertiti in indirizzi assoluti dal loader (codice rilocabile)
- **dinamicamente**
  - a tempo di esecuzione: durante l'esecuzione un processo puo' essere spostato da un'area all'altra

Caricamento/collegamento dinamico

Obiettivo: ottimizzazione della memoria

Caricamento dinamico

- in alcuni casi e' possibile caricare in memoria una funzione/procedura a runtime solo quando avviene la chiamata
- loader di collegamento rilocabile: carica e collega dinamicamente la funzione al programma che la usa
- la funzione puo' essere usata da piu' processi simultaneamente. Problema di visibilita' --> compito SO e' concedere/controllare:
  - l'accesso di un processo allo spazio di un altro processo
  - l'accesso di piu' processi agli stessi indirizzi

Overlay

In generale, la memoria disponibile non puo' essere sufficiente ad acogliere codice e dati di un processo.

Soluzione a **overlay** mantiene in memoria istruzioni e dati:

- che vengono utilizzati piu' frequentemente
- che sono necessari nella fase corrente

Codice e dati di un processo vengono suddivisi in **overlay** che vengono caricati e scaricati dinamicamente (dal gestore di overlay, di solito esterno al SO)

Overlay: esempio

Assembler a 2 passi: produce l'eseguibile di un programma assembler, mediante 2 fasi sequenziali

1. Creazione della tabella dei simboli (passo 1)
2. Generazione dell'eseguibile (passo 2)

4 componenti distinte nel codice assembler:

- Tabella dei simboli (ad es. dim 20KB)
- Sottoprogrammi comuni ai due passi (ad es. 30KB)
- Codice passo 1 (ad es. 70KB)
- Codice passo 2 (ad es. 80KB)

--> spazio richiesto per l'allocazione integrale dell’assembler e' quindi 200KB

Tecniche di allocazione memoria centrale

Come vengono allocati codice e dati dei processi in memoria centrale?

Varie tecniche

- Allocazione Contigua
  - a partizione singola
  - a partizioni multiple
- Allocazione non contigua
  - paginazione
  - segmentazione

Allocazione contigua a partizione singola

Primo approccio molto semplificato: la parte di memoria disponibile per l'allocazione dei processi di utente non e' partizionata

- un solo processo alla volta puo' essere allocato in memoria: non c'e' multiprogrammazione

Di solito:

- SO richiede nella memoria bassa [0, max]
- necessita' di proteggere codice e dati di SO da accessi di processi utente:
    - uso del registro rilocazione (RL=max+1) per garantire la correttezza degli accessi

Allocazione contigua: partizioni multiple

Multiprogrammazione --> necessita' di proteggere codice e dati di ogni processo

Partizioni multiple: ad ogni processo caricato viene associata un'area di memoria distinta (partizione)

- partizioni fisse
- partizioni variabili

Partizioni fisse (MFT, multiprogramming with Fixed number of Tasks): dim di ogni partizione fissata a priori

- ogni partizione ha dimensione prefissata, eventualmente diversa da partizione a partizione
- quando un processo viene schedulato, SO cerca una partizione libera di dim sufficiente

Problemi:

- **frammentazione interna**: sottoutilizzo della partizione
- **grado di multiprogrammazione limitato** al numero di partizioni
- **dim massima** dello spazio di indirizzamento di un processo limitata da dim della partizione piu' estesa

Partizioni variabili

Partizioni variabili (MVT, Multiprogramming with Variable number of Tasks): ogni partizione allocata dinamicamente e dimensionata in base a dim processo da allocare

- quando un processo viene schedulato, SO cerca un'area sufficientemente grande per allocarvi dinamicamente la partizione associata

Vantaggi (rispetto a MFT):

- elimina frammentazione interna (ogni partizione e' della esatta dimansione del processo)
- grado di multiprogrammazione variabile
- dimansione massima dello spazio di indirizzamento di ogni processo limitata da dim spazio fisico

Problemi:

- scelta dell'area in cui allocare: best fit, worst fit, first fit, ...

- frammentazione esterna - man mano che si allocano nuove partizioni, la memoria libera e' sempre piu' frammentata
    - --> necessita' di compattazione periodica

Partizioni & protezione

Protezione realizzata a livello HW mediante:

- registro di rilocazione RR
- registro limite RL

Ad ogni processo e' associata una coppia di valori $<V_{RR}, V_{RL}>$. Quando un processo P viene schedulato, il dispatcher carica RR e RL con i valori associati al processo $<V_{RR}, V_{RL}>$

Paginazione

Allocazione contigua a partizioni multiple: il problema principale e' la frammentazione esterna.

Allocazione non contigua --> paginazione

- eliminazione frammentazione esterna
- riduzione forte di frammentazione interna

Idea di base: partizionamento spazio fisico di memoria in pagine (frame) di dm costante e limitata (ad es. 4KB) sulle quali mappare porzioni di processi da allocare.

- **Spazio fisico**: insieme di frame di D_f costante prefissata
- **Spazio logico**: insieme di pagine di dm uguale a D_f

Ogni pagina logica di un processo caricato in memoria viene mappata su una pagina fisica in memoria centrale

Vantaggi

- Pagine logiche contigue possono essere allocate su pagine fisiche non contigue: non c'e' frammentazione esterna
- Le pagine sono di dim limitata: frammentazione interna per ogni processo limitata dalla dimensione del frame
- E' possibile caricare in memoria un sottoinsieme delle pagine logiche di un processo

Supporto HW a Paginazione

Struttura dell'indirizzo logico (m bit):**

p	d

- p - numero di pagina logica (m-n bit)
- d - offset della cella rispetto all'inizio della pagina (n bit)

Hp: indirizzi logici *m* bit (*n* bit per offsett, e *m-n* per la pagina)

- dim massima dello spaio logico di indirizzamento --> 2^m
- dim della pagina --> 2^n
- numero di pagine --> 2^{m-n}

Struttura dell'indirizzo fisico:

f	d

- f - numero d frame (pagina fisica)
- d - offset della cella rispetto all'inizio del frame

Binding tra indirizzi logici e fisici puo' essere realizzato mediante **tabella delle pagine** (associata al processo):

- a ogni pagina logica associa la pagina fisica verso la quale e' realizzato il mapping

Realizzazione della tabella delle pagine

Problemi da affrontare

- tabella puo' essere molto grande
- traduzione (ind. logico --> ind. fisico) deve essere il piu' veloce possibile

Varie soluzioni

1. Su registri di CPU
   - accesso veloce
   - cambio di contesto pesante
   - dimensioni limitate della tabella
2. In memoria centrale: registro PageTableBaseRegister (PTBR) memorizza collocazione tabella in memoria
   - 2 accessi in memoria per ogni operazione (load, store)
3. Uso di cache: (Translation Look-aside Buffers, TLB) per velocizzare l'accesso

Translation Look-aside Buffers (TLB)

- Tabella delle pagine e' allocata in memoria centrale

- Una parte della tabella delle pagine (di solito, le pagine accedute piu' d frequente o piu' di recente) e' copiata in cache: TLB

Se la coppia (p, f) e' gia' presente in cache l'accesso e' veloce; altrimenti SO deve trasferire la coppia richiesta dalla tabella delle pagine (in memoria centrale) in TLB

Gestione TLB

- TLB inizialmente vuoto
- mentre l'esecuzione procede, viene gradualmente riempito con indirizzi pagine gia' accedute

HIT-RATIO: percentuale di volte che una pagina viene trovata in TLB

- dipende dalla dimensione del TLB (Intel486: 98%)

Paginazione & protezione

La tabella delle pagine

- ha dimensione fissa
- non sempre viene utilizzata completamente

Come distinguere gli elementi significativi da quelli non utilizzati?

- **Bit di validita'**: ogni elemento contiene un bit
  - se e' a 1, entry valida (pagina appartiene allo spazio logico del processo)
  - se e' 0, entry non valida
- **Page Table Length Register**: registro che contiene il numero degli elementi validi nella tabella delle pagine

In aggiunta, per ogni entry della tabella delle pagine, possono esserci uno o piu' nit di protezione che esprimono le modalita' di accesso alla pagina (Es. read-only)

Paginazione a piu' livelli

Lo spazio logico di indirizzamento di un processo puo' essere molto esteso:

- elevato numero di pagine
- tabella delle pagine di grandi dimensioni

Ad esempio:

- per ipotesi, indirizzi di 32 bit --> spazio logico di 4GB dimansioone pagina 4KB (2^{12})
- la tabella delle pagine dovrebbe contenere $2^{32}/2^{12}$ elementi --> 2^{20} elementi circa (1M)

Paginazione a piu' livelli: allocazione non contigua anche delle pagine --> si applica ancora la tecnica di paginazione alla tabella delle pagine!

Vantaggi:

- possibilita' di indirizzare spazi logici di dimensione elevate riducendo i problemi di allocazione delle tabelle
- possibilita' di mantenere in memoria soltanto le pagine della tabella che servono

Svantaggio:

- tempo di accesso piu' elevato: per tradurre un indirizzo logico sono necessari piu' accessi in memoria (ad esempio, 2 livelli di paginazione --> 2 accessi)

Tabella delle pagine invertita

Per limitare l'occupazione di memoria, in alcuni SO si usa un'unica struttura dati globale che ha un elemento per ogni frame: **tabella delle pagine invertita**

Ogni elemento della tabella delle pagine invertita rappresenta un frame (indirizzo pari alla posizione della tabella) e, in caso di frame allocato, contiene:

- **pid**: identificatore del processo a cui e' assegnato il frame
- **p**: numero di pagina logica

Per tradurre l'indirizzo logico <pid, p, d>:

- ricerca nella tabella dell'elemento che contiene la coppia (pid, p) --> l'indice dell'elemento trovato rappresenta il numero del frame allocato alla pagina logica p

Problemi:

- tempo di ricerca nella tabella invertita
- difficolta' di realizzazione della condivisione di codice tra processi (**rientranza**): come associare un frame a piu' pagine logiche di processi diversi?

Organizzazione di memoria in segmenti

La segmentazione si basa sul partizionamento dello spazio logico degli indirizzi di un processo in parti (segmenti), ognuna caratterizzata da nome e lunghezza

- Divisione semantica per funzione: ad esempio
  - codice
  - stack
  - dati
  - heap
- Non e' stabilito un ordine tra i segmenti
- Ogni segmento allocato in memoria in modo contiguo

- Ad ogni segmento SO associa un intero attraverso il quale lo si puo' riflettere

Segmentazione

Struttura degli indirizzi logici: ogni indirizzo e' costituito dalla coppia <segmento, offset>

- segmento: numero che individua il segmento nel sistema
- offset: posizione cella all'interno del segmento

Supporto HW alla segmentazione

Tabella dei segmenti: ha una entry per ogni segmento che ne descrive l'allocazione in memoria fisica mediante la coppia <base, limite>

- base: indirizzo prima cella del segmento nello spazio fisico
- limite: indica la dimensione del segmento

Realizzazione della tabella dei segmenti

Tabella globale: possibilita' di dimensioni elevate

Realizzazione

- Su registri di CPU
- In memoria, mediante registri base (Segment Table Base Register, STBR) e limite (Segment Table Length Register, STLR)
- Su cache (solo l'insieme dei segmenti usati piu' recentemente)

Segmentazione

Estensione della tecnica di allocazione a partizioni variabili

- partizioni variabili: 1 segmento/processo
- segmentazione: piu' segmenti/processo

Problema principale:

- come nel caso delle partizioni variabili, **frammentazione esterna**

Soluzione: allocazione dei segmenti con tecniche

- **best fit**
- **worst fit**
- **...**

Segmentazione paginata

Segmentazione e paginazione possono essere combinate (ad esempio Intel x86):

- spazio logico segmentato (Specialmente per motivi di protezione)
- ogni segmento suddiviso in pagine

Vantaggi:

- eliminazione della frammentazione esterna (ma introduzione di frammentazione interna ...)
- non necessario mantenere in memoria l'intero segmento, ma e' possibile caricare soltanto le pagine necessarie

Strutture dati:

- tabella dei segmenti
- una tabella delle pagine per ogni segmento

Ad esempio, segmentazione in Linux

Linux adotta una gestione della memoria basata su segmentazione paginata

Vari tipi di segmento:

- code (kernel, user)
- data (kernel, user)
- task state segments (registri dei processi per il cambio di contesto)
- ...

I segmenti sono paginati con paginazione a tre livelli

Memoria virtuale

La dimensione della memoria puo' rappresentare un vincolo importante, riguardo a

- dimensione dei processi
- grado di multiprogrammazione

Puo' essere desiderabile un sistema di gestione della memoria che:

- consenta la presenza di **piu' processi** in memoria (ad es. partizioni multiple, paginazione e segmentazione), indipendentemente dalla dimensione dello spazio disponibile
- svincoli il grado di multiprogrammazione dalla dimensione effettiva della memoria

Con le tecniche viste finora:

- l'intero spazio logico di ogni processo e' allocato in memoria

- overlay, caricamento dinamico: si possono allocare/deallocare parti dello spazio di indirizzi --> a carico del programmatore

Memoria Virtuale: e' un metodo di gestione della memoria che consente l'esecuzione di processi non completamente allocati in memoria principale

Vantaggi:

- **dimensione dello spazio logico degli indirizzi non vincolata** all'estensione della memoria
- **grado di multiprogrammazione indipendente** dalla dimensione della memoria fisica
- **efficienza**: caricamento di un processo e swapping hanno un costo piu' limitato (meno I/O)
- **astrazione**: il programmatore non deve preoccuparsi dei vincoli relativi alla dimensione della memoria

Memory Management Unit

- Nei calcolatori senza memoria virtuale, una "move reg, 1000" provoca una lettura o scrittura di un byte all'indirizzo fisico 1000
- Con la memoria virtuale l'indirizzo viene inviato dalla CPU alla MMU che trasforma l'indirizzo logico in uno fisico
- Gli indirizzi (generati usando registri base, registri indici, etc..) si chiamano indirizzi virtuali (logici) e costituiscono lo spazio degli indirizzi virtuali

Paginazione su richiesta

Di solito la memoria virtuale e' realizzata mediante tecniche di paginazione su richiesta:

- **tutte le pagine di ogni processo possiedono una memoria di massa**; durante l'esecuzione alcune di esse vengono trasferite, all'occorrenza, in memoria centrale

Pager: modulo del SO che realizza i trasferimenti delle pagine da/verso memoria secondaria/centrale ("swapper" di pagine)

- paginazione su richiesta (o "su domanda"): pager lazy ("pigro") trasferisce in memoria centrale una pagina soltanto se ritenuta necessaria

Esecuzione di un processo puo' richiedere swap-in del processo

- **swapper**: gestisce i trasferimenti di interi processi (mem. centrale <--> mem. secondaria)
- **pager**: gestisce i trasferimenti di singole pagine

Prima di eseguire swap-in di un processo:

- pager puo' prevedere le pagine di cui (probabilmente) il processo avra' bisogno inizialmente --> caricamento

HW necessario:

- tabella delle pagine (con PTBR, PTLR, e/o TLB, ...)
- **memoria secondaria** e strutture necessarie per la sua gestione (uso di dischi veloci)

Quindi in generale, una pagina dello spazio logico di un processo:

- puo' essere **allocata in memoria centrale**
- puo' essere **in memoria secondaria**

Come distinguere i due casi?

La tabella delle pagine contiene **bit di validita'**:

- se la pagina e' presente in memoria centrale
- se e' in memoria secondaria oppure e' invalida --> interruzione al SO (**page fault**)

Trattamento page fault

Quando kernel SO riceve l'interruzione dovuta al page fault

1. Salvataggio del contesto di esecuzione del processo (registri, stato, tabella delle pagine)
2. Verifica del motivo del page fault (tramite una tabella interna al kernel)
    - riferimento **illegale (violazione delle politiche di protezione**) --> terminazione del processo
    - riferimento **legale**: la pagina in memoria e' secondaria
3. Copia della pagina in un frame libero
4. Aggiornamento della tabella delle pagine
5. Ripristino del processo: esecuzione dell'istruzione interrotta dal page fault

Paginazione su richiesta: sovrallocazione

In seguito a un page fault:

- se e' necessario caricare una pagina in memoria centrale, puo' darsi che non ci siano frame liberi

Sovrallocazione

Soluzione

--> **sostituzione** di una pagina P_{vitt} (vittima) allocata in memoria con la pagina P_{new} da caricare:

1. Individuazione della vittima P_{vitt}
2. Salvataggio di P_{vitt} su disco
3. Caricamento di P_{new} nel frame liberato
4. Aggiornamento tabelle

5. Ripresa del processo

Sostituzione di pagine

In generale, la sostituzione di una pagina puo' richiedere **2 trasferimenti da/verso il disco:**

- per scaricare la vittima
- per caricare la pagina nuova

Pero' e' possibile che la vittima non sia stata modificata rispetto alla copia residente sul disco, ad esempio:

- pagine del codice (read-only)
- pagine contenenti dati che non sono stati modificati durante la permanenza in memoria

In questo caso la copia della vittima sul disco puo' essere evitata:
--> introduzione del **bit di modifica (dirty bit)**

Dirty bit

Per rendere piu' efficiente il trattamento del page fault in caso di sovrallocazione

- si introduce in ogni elemento della tabella delle pagine un bit di modifica (**dirty bit**)
  - se settato, la pagina ha subito almeno un aggiornamento in memoria da quando e' caricata la memoria
  - se a 0, la pagina non e' stata modificata
- algoritmo di sostituzione esamina il bit di modifica della vittima:
  - esegue swap-out della vittime solo se il dirty but e' settat

Algoritmi di sostituzione della pagina vittima

La finalita' di ogni algoritmo di sostituzione e' sostituire quella pagina la cui probabilita' di essere accedute a breve termine e' bassa

Algoritmi

- **LFU** (Leatest Frequently Used): sostituita la pagina che e' stata usata meno frequentemente (in un intervallo di tempo prefissato)
  - e' necessario associare un contatore degli accessi ad ogni pagina
  - la vittima e' quella con minimo valore del contatore
- **FIFO**: sostituita la pagina che e' da piu' tempo caricata in memoria (indipendentemente dal suo uso)
  - e' necessario memorizzare la cronologia dei caricamenti in memoria

- **LRU** (Least Recently Used): di solito preferibile per principio di localita', viene sostituita la pagina che e' stata usata meno recentemente
  - e' necessario registrare la sequenza degli accessi alle pagine in memoria
  - overhead, dovuto all'aggiornamento della sequenza degli accessi per ogni accesso in memoria

Implementazione LRU: necessario registrare la sequenza temporale degli accesssi

Soluzioni

- **Time** sharing: l'elemento della tabella delle pagine contiene un campo che rappresenta l'istante dell'ultimo accesso alla pagina
  - Costo della ricerca della pagina vittima
- **Stack**: struttura dati di tipo stack in cui ogni elemento rappresenta una pagina; l'accesso a una pagina provoca lo spostamento dell'elemento corrispondente al top dello stack --> **bottom contiene la pagina LRU**
  - gestione puo' essere costosa, ma non c'e' overhead di ricerca

Algoritmi di sostituzione: LRU approssimato

Spesso si utilizzano versioni semplificate di LRU introducendo, al posto della sequenza degli accessi, un bit di uso associato alla pagina:

- al momento del caricamento e' inizializzato a 0
- quando la pagina viene acceduta, viene settato
- periodicamente, i bit di uso vengono resettati

--> viene sostituita una pagina con bit di uso==0; il criterio di scelta, ad esempio, potrebbe inoltre considerare il dirty bit:

- tra tutte le pagine non usate di recente (bit di uso0), ne viene scelta una non aggiornata (dirty bit0)

Localita' dei programmi

Si e' osservato che un processo, in una certa fase di esecuzione:

-
  - usa solo un sottoinsieme relativamente piccolo delle sue pagine logiche
    - sottoinsieme delle pagine effettivamente utilizzate varia lentamente nel tempo
- **Localita' spaziale**
  - alta probabilita' di accedere a locazioni vicine (nello spazio logico/virtuale) a locazioni appena accedute (ad esempio, elementi di un vettore, codice sequenziale, ...)
- **Localita' temporale**

- alta probabilita' di accesso a locazioni accedute di recente (ad esempio cicli)

Working set

In alternativa alla paginazione su domanda, tecniche di gestione della memoria che si basano su pre-paginazione:

- si prevede il seti di pagine di cui il processo da caricare ha bisogno per la proissima fase di esecuzione
  working set

Working set puo' essere individuato in base a criteri di **localita' temporale**

Dato un intero **Δ**, il working set di un processo P (nell'istante t) e' l'insieme di pagine **Δ**(t) indirizzate da P nei piu' recenti **Δ** riferimenti.
Δ definisce la "finestra" del working set

Prepaginazione con working set

- Caricamento di un processo consiste nel caricamento su working set iniziale
- SO mantiene working set di ogni processo aggiornandolo dinamicamente, in base al principio di localita' temporale:
  - all'istante t vengono mantenute le pagine usate dal processo nell'ultima finestra **Δ** (t)
  - le altre pagine (esterne a **Δ** (t)) possono essere sostituite

**
Vantaggio**

- riduzione del numero di page fault

Un esempio: gestione della memoria in UNIX (prime versioni)

In UNIX spazio logico segmentato:

- nelle prime versioni (prima di BSDv3), allocazione contigua dei segmenti
  - segmentazione pura
  - non c'era memoria virtuale
- in caso di difficolta' di allocazione dei processi --> **swapping dell'intero spazio degli indirizzi**
- **condivisione codice**: possibilita' di evitare trasferimenti di codice da memoria secondaria a memoria centrale --> minor overhead di swapping

Tecnica di **allocazione contigua** dei segmenti:

- **first fit** sia per l'allocazione in memoria centrale che in memoria secondaria (swap-out)

Problemi

- frammentazione esterna
- stretta influenza dim spazio fisico sulla gestione dei processi in multiprogrammazione
- crescita dinamica dello spazio --> possibilita' di riallocazione di processi gia' caricati in memoria

UNIX: swapping

In assenza di memoria virtuale, **swapper** ricopre un ruolo chiave per la **gestione delle contese di memoria** da parte dei diversi processi:

- periodicamente (ad esempio nelle prime versioni ogni 4s) lo swapper viene attivato per provvedere (eventualmente) a **swap-in** e **swap-out** di processi
  - **swap-out**:
    - processi inattivi (sleeping)
    - processi "ingombranti"
    - processi da piu' tempo in memoria
  - **swap-in**:
    - processi piccoli
    - processi da piu' tempo **swapped**

La gestione della memoria in UNIX (versioni attuali)

Da BSDv3 in poi:

- **segmentazione paginata**
- memoria virtuale tramite paginazione su richiesta

L'allocazione di ogni segmento non e' contigua:

- si risolve il problema della frammentazione esterna
- frammentazione interna trascurabile (pagine di dimensioni piccole)

La gestione della memoria in UNIX (versioni attuali)

Paginazione su richiesta

- **pre-paginazione**: uso dei frame liberi per per-caricare pagine non strettamente necessarie. Quando viene un page fault, se la pagina e' gia' in un frame libero, basta soltanto modificare:
  - tabella delle pagine

    - lista dei frame liberi
- **core map**: struttura dati interna al kernel che descrive lo **stato di allocazione dei frame** e che viene consultata in caso di page fault

UNIX: algoritmo di sostituzione

LRU modificato o algoritmo di seconda chance (BSDv4.3 Tahoe)
ad ogni pagina viene associato un bit di uso:

- al momento del caricamento e' inizializzato a 0
- quando la pagina viene acceduta, viene settato a 1
- nella fase di ricerca di una vittima, vengono esaminati i bit di uso di tutte le pagine in memoria
    - se una pagina ha il bit di uso a 1, viene posto a 0
    - se una pagina ha il bit di uso a 0, viene selezionata come vittima

Sostituzione della vittima:

- pagina viene **resa invalida**
- frame selezionato viene inserito **nella lista dei frame liberi**
    - se c'e' **dirty bit**:
        - **solo se dirty bit=1** --> pagina va copiata in memoria secondaria
        - se non c'e' **dirty bit** --> pagina va sempre copiata in memoria secondaria

L'algoritmo di sostituzione viene eseguito da pager
pagedaemon (pid=2)

UNIX: sostituzione delle pagine

Scaricamento di pagine (sostituzione) attivato quando numero totale di frame liberi e' ritenuto insufficiente (minore del valore **lotsfree**)

Parametri

- **lotsfree**: numero minimo di frame liberi per evitare sostituzioni di pagine
- **minfree**: numero minimo di frame liberi necessari per evitare swapping dei processi
- **desfree**: numero desiderato di frame liberi

lotsfree > desfree > minfree

UNIX: scheduling, paginazione e swapping

Scheduler attiva l'algoritmo di sostituzione se

- il numero di frame liberi < lotsfree

Se il sistema di **paginazione e' sovraccarico**, ovvero:

- numero di frame liberi < minfree
- numero medio di frame liberi nell'unita' di tempo < desfree

--> scheduler attiva swapper (al massimo ogni secondo)

SO evita che pagedaemon usi piu' del 10% del tempo totale di CPU: attivazione (al massimo) ogni 250ms

Gestione della memoria in Linux

- Allocazione su **segmentazione paginata**
- Paginazione **a piu' (2 o 3) livelli**
- Allocazione contigua dei moduli di codice caricati dinamicamente
- **Memoria virtuale, senza working set**

Linux: organizzazione della memoria fisica

Alcune aree riservate a scopi specifici:

- **Area codice kernel**: pagine di quest'area sono locked (non subiscono paginazione)
- **Kernel cache**: heap del kernel (locked)
- **Area moduli gestiti dinamicamente**: allocazione mediante algoritmo buddy list (allocazione contigua dei songoli moduli)
- **Buffer cache**: gestione I/O su dispositivi a blocchi
- **Inode cache**: copia degli inode utilizzati recentemente
- **Page cache**: pagine non piu' utilizzate in attesa di sostituzione

Il resto della memoria e' utilizzato per i processi utente

Linux: spazio di indirizzamento

Ad ogni processo Linux possono essere allocati 4GB di memoria centrale (in caso di sistema a 32 bit):

- 3GB al massimo possono essere utilizzati per lo **spazio di indirizzamento virtuale**
- 1GB **riservato al kernel**, accessibile quando il processo esegue in kernel mode
- con architettura a 64 bi, spazio di indirizzamento >= 1TB, paginazione a 4 livelli

Spazio di indirizzamento di ogni processo puo' essere suddiviso in un insieme di **regioni omogenee e contigue**

- ogni regione e' costituita da una sequenza di pagine accomunate dalle stesse caratteristiche di protezione e di paginazione
- ogni pagina ha una dimensione costante (4KB su architettura Intel)

Linux: page-fetching e sostituzione

- NON viene utilizzata la tecnica del **working set**
- viene mantenuto un **insieme di pagine libere** che possano essere utilizzate dai processi (**page cache**)
- analogamente a UNIX, una volta al secondo:
  - viene controllato che ci siano **sufficienti pagine libere**
  - altrimenti, viene **liberata una pagina occupata**