Memoria centrale

Pietro Braione

Reti e Sistemi Operativi – Anno accademico 2021-2022

Obiettivi

- Spiegare la differenza tra indirizzo logico e fisico, e il ruolo della MMU nella traduzione da indirizzi logici ad indirizzi fisici
- Comprendere le strategie first-fit, worst-fit e best-fit per allocare la memoria in modo contiguo
- Spiegare la differenza tra frammentazione interna ed esterna
- Capire come gli indirizzi logici vengono tradotti in indirizzi fisici in un sistema con paginazione
- Confrontare i diversi tipi di tabelle delle pagine: gerarchiche, di tipo hash, e invertite

Ricapitoliamo un po’ di background…

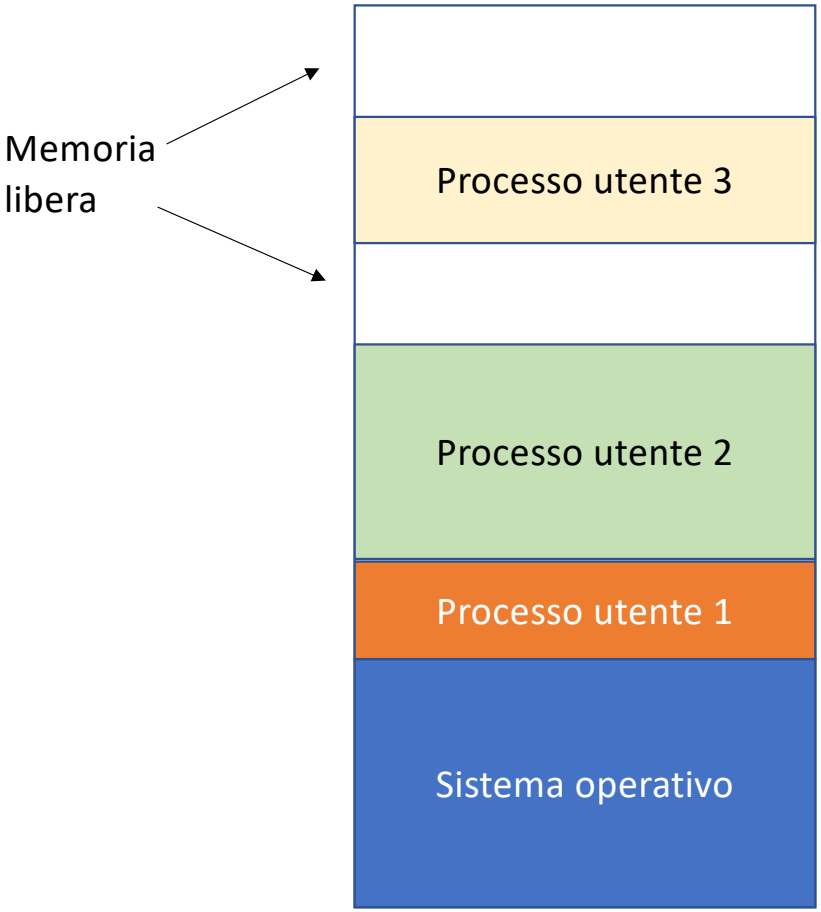
- Le aree di memoria che i processori possono usare direttamente per l’esecuzione dei programmi sono solo la memoria centrale ed i registri
- Pertanto un programma deve essere portato dal disco in memoria centrale perché possa essere eseguito, e il programma deve avere anche sufficiente memoria centrale per memorizzare i risultati della computazione
- La memoria centrale «vede» solo un flusso di indirizzi (richieste di lettura o scrittura) proveniente dal bus di sistema, e non è consapevole di chi ha generato tale flusso
- Le operazioni sui registri richiedono un ciclo di clock (o anche meno)
- Viceversa, le operazioni sulla memoria richiedono molti cicli di clock, anche diverse centinaia, causando uno stallo (stall) del processore, durante il quale un core multithread può eseguire un altro thread hardware
- Per aumentare l’efficienza degli accessi in memoria vengono utilizzati diversi livelli di memorie cache tra processore e memoria centrale

Il problema dell'allocazione della memoria

- In sostanza, se il sistema operativo vuole mandare in esecuzione un processo, deve caricare la sua **immagine** (codice + dati statici + stack + heap) *nella sua interezza* in memoria centrale
- In un sistema multiprogrammato, più immagini di più processi sono caricate contemporaneamente nella memoria centrale
- Il sistema operativo deve, pertanto, allocare porzioni di memoria centrale ai diversi processi in funzione delle necessità di tali processi
- Questo pone diversi problemi:
  - Che strategie di allocazione possiamo adottare?
  - Come proteggiamo la memoria del sistema operativo dai processi, e quella di ogni processo da ogni altro processo?
  - Se l'immagine di un programma può essere caricata, in momenti diversi, ad indirizzi fisici diversi, come fanno le istruzioni del programma a referenziare una certa locazione di memoria?

Allocazione contigua

- È la strategia più semplice di allocazione della memoria in un sistema multiprogrammato
- La memoria centrale è partizionata in due zone, una per il sistema operativo e una per i processi utente
- Ogni processo utente occupa un'area contigua di memoria nella partizione dei processi utente, e in quell'area viene caricata la sua immagine

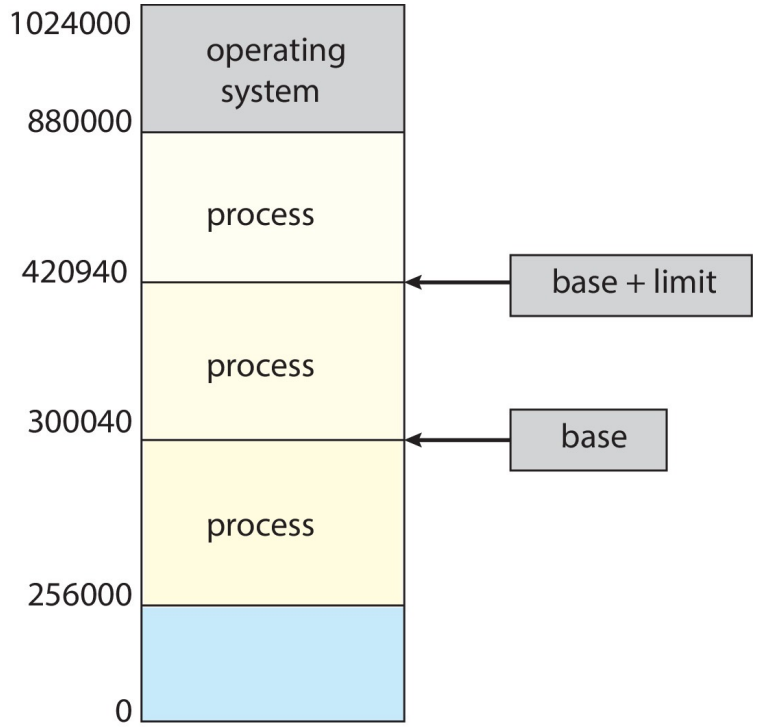


Protezione

- È necessario che la memoria centrale assegnata ad un processo sia protetta dalle operazioni sulla memoria effettuate da un altro processo:
  - Per robustezza in caso di errori software
  - Ma anche per sicurezza (un processo potrebbe osservare o corrompere i dati di un altro)
- Questo tipo di protezione viene assicurato da meccanismi hardware, che possono essere di diversi tipi

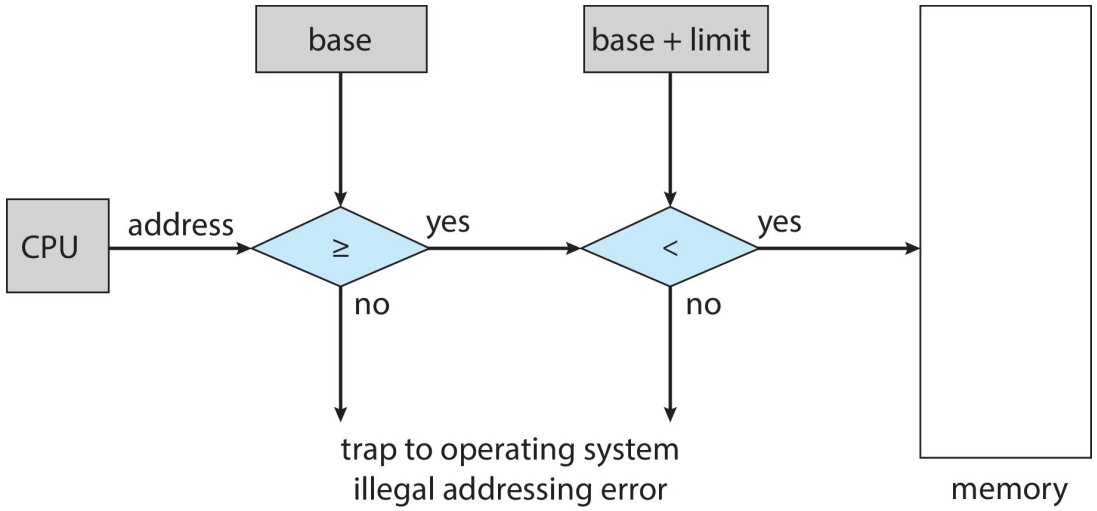
Protezione con registro base e limite

- Il meccanismo di protezione più semplice è dotare il processore di due registri, un registro **base** ed un registro **limite**
- Il registro base contiene il più piccolo indirizzo della memoria fisica che il processo corrente ha il permesso di accedere
- Il registro limite determina la dimensione dell'intervallo ammesso



Protezione hardware degli indirizzi

- I registri base e limite possono essere impostati solo in modalità kernel
- In modalità utente il processore proibisce tutte le operazioni di lettura/scrittura fuori dall'intervallo individuato dai registri base e limite
- Nel caso in cui venga generato un indirizzo fuori dall'intervallo, l'indirizzo non viene messo sul bus e viene generata un'eccezione



Associazione degli indirizzi

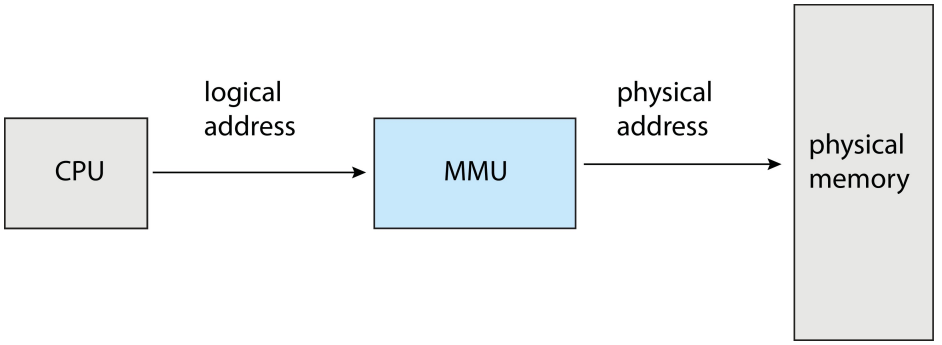
- C’è però un problema legato al caricamento di un programma in memoria
- In un sistema multiprogrammato il sistema operativo deve poter caricare uno stesso programma, in momenti diversi, in diverse aree di memoria
- Come fa, pertanto, un programma a far riferimento ad una certa locazione di memoria?
- Notiamo che nei programmi gli indirizzi di memoria hanno rappresentazioni diverse a seconda della fase di elaborazione del programma stesso:
  - Nel codice sorgente, se il linguaggio è ad alto livello, non si usano indirizzi di memoria
  - Nella compilazione viene prodotto un file oggetto con indirizzi **rilocabili:** di solito essi sono spiazzamenti relativi all’inizio della zona di memoria dove si presume che il modulo sarà caricato
  - Il linker e/o il loader devono far corrispondere tali indirizzi ad indirizzi assoluti quando, alla fine, il programma sarà caricato in memoria
- Queste operazioni di «traduzione» di una forma di indirizzo in un’altra sono dette di **associazione** (**binding**)

Associazione degli indirizzi: varianti

- Il binding può essere fatto in tre momenti diversi:
  - **In compilazione**, se conosciamo a priori l'indirizzo a partire dal quale il programma verrà caricato: in tal caso il compilatore/linker effettua il binding e genera **codice assoluto**. Se cambia l'indirizzo di caricamento il codice va ricompilato
  - **In caricamento**, se se non conosciamo a priori l'indirizzo a partire dal quale il programma verrà caricato, e il programma non verrà spostato in memoria durante la sua esecuzione: in tal caso il compilatore/linker genera codice rilocabile e il loader effettua il binding al momento dell'esecuzione.
  - **In esecuzione**, se se non conosciamo a priori l'indirizzo a partire dal quale il programma verrà caricato, e il programma può essere spostato in memoria *durante* l'esecuzione: in tal caso occorre il supporto di hardware speciale nel processore.
- La terza soluzione è impiegata dalla maggior parte dei sistemi operativi general-purpose, e la analizzeremo nel seguito

Spazi di indirizzi logici e fisici

- Gli indirizzi generati dalla CPU quando esegue un programma sono detti **indirizzi logici**
- Gli indirizzi che arrivano alla memoria centrale sono detti **indirizzi fisici**
- Il binding in fase di esecuzione è l'unico metodo nel quale indirizzi logici e fisici differiscono
- La **memory management unit** (MMU) è il dispositivo hardware che traduce indirizzi logici in indirizzi fisici
- La MMU interviene solo in modalità utente: In modalità kernel gli indirizzi generati dalla CPU sono direttamente indirizzi fisici
- (Quindi, se il sistema operativo deve accedere alla memoria di un processo, deve tradurre «manualmente» gli indirizzi logici del processo in fisici)

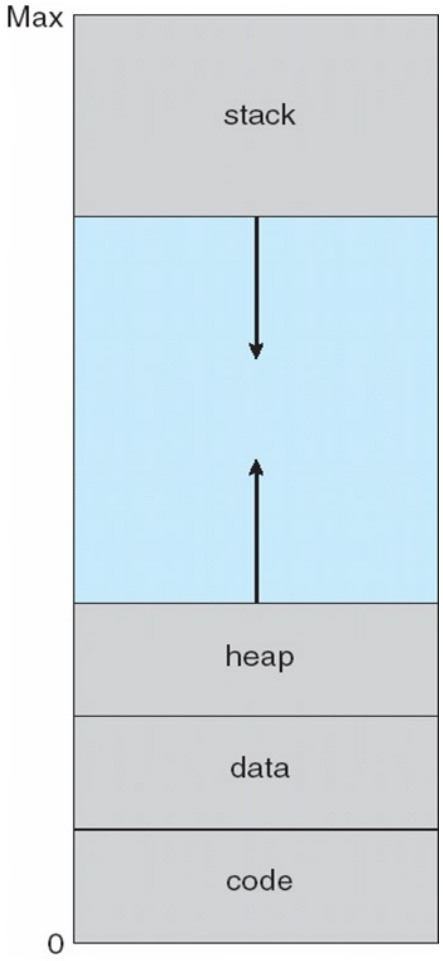


MMU con registro di rilocazione

- La MMU più semplice è quella con registro di rilocazione
- È una variazione dello schema con registri base e limite, ma il registro base è ora chiamato **registro di rilocazione**
- L'indirizzo fisico è ottenuto sommando all'indirizzo logico il valore del registro di rilocazione
- In tal modo i programmi hanno l'illusione di avere uno spazio di indirizzi (logico) che va dall'indirizzo 0 a un indirizzo massimo pari al valore contenuto nel registro limite

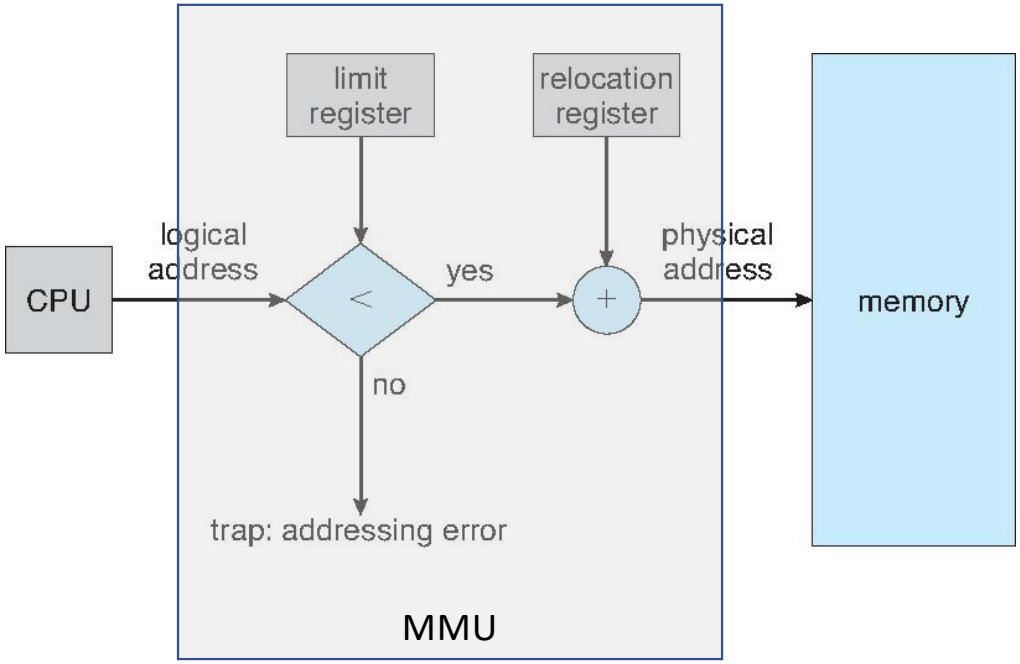
Spazio di indirizzamento virtuale

- Lo **spazio di indirizzamento virtuale** (**virtual address space**, o **VAS**) è lo spazio di indirizzi logici di un processo
- Si estende da un certo indirizzo logico (tipicamente 0) ad un indirizzo massimo
- Di solito organizzato con codice e dati statici negli indirizzi bassi, quindi heap che cresce verso indirizzi alti e stack da indirizzo massimo verso indirizzi bassi, con buco nel mezzo:
  - Permette uno spazio di indirizzi sparso
  - Permette di allocare librerie dinamiche nel buco tra stack e heap
- Se basato sulla paginazione (vedere più avanti) permette di condividere memoria tra processi:
  - Librerie condivise (read-only)
  - Memoria condivisa (IPC)
  - Fork rapido dei processi (con tecnica copy-on-write)
- Ne parleremo in dettaglio quando introdurremo la memoria virtuale (demand paging)



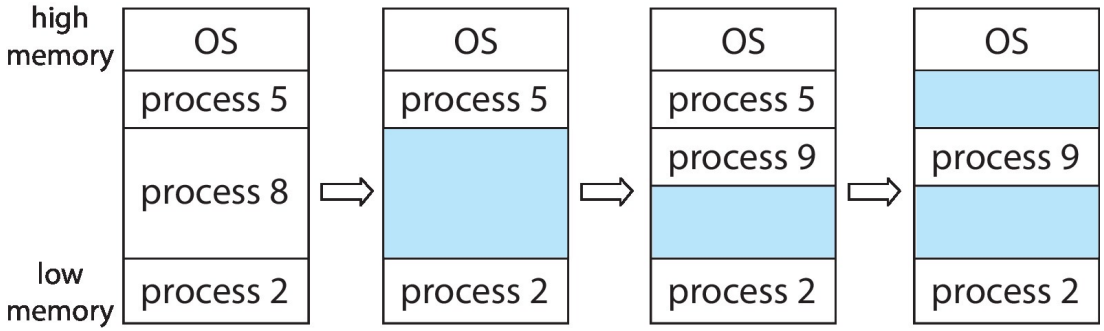
Protezione della memoria nell'allocazione contigua

- L'allocazione contigua è facilmemte realizzabile con la MMU con registro di rilocazione
- Nel registro base viene caricato l'indirizzo più basso dell'area contigua di memoria assegnata al processo
- Nel registro limite viene caricata la dimensione di tale area di memoria
- Se l'indirizzo logico supera tale valore, viene generata un'interruzione (intercettata dall'OS, che di solito interrompe il processo)



Allocazione contigua a partizioni variabili (1)

- Ogni processo ottiene una partizione di memoria distinta
- Schema a **partizioni variabili**: partizione di dimensione pari alla memoria necessaria al processo
- Un buco è una regione di memoria libera contigua, ed ogni processo riceve la sua partizione di memoria da un buco abbastanza grande da contenerla
- Quando un processo termina libera la sua partizione creando un nuovo buco, e buchi adiacenti sono uniti



Allocazione contigua a partizioni variabili (2)

- Il sistema operativo mantiene una lista dei buchi disponibili sparsi nella memoria centrale
- Alla creazione di un nuovo processo il sistema operativo sceglie un buco dal quale prendere la memoria necessaria ad esso secondo una strategia:
  - **First-fit**: sceglie il primo buco sufficientemente grande da contenere l'immagine del processo
  - **Best-fit**: sceglie il buco più piccolo
  - **Worst-fit**: sceglie il buco più grande
- First fit e best-fit sono sperimentalmente migliori in quanto a tempo ed efficienza

Frammentazione (1)

- **Frammentazione esterna**: la memoria libera è sufficiente per la creazione di un nuovo processo, ma è sparsa tra buchi non contigui troppo piccoli
- **Frammentazione interna**: se la memoria allocata ad un processo è più grande della memoria necessaria, una partizione può contenere memoria inutilizzata
- Regola del 50%: lo spazio inutilizzabile tende ad essere circa il 50% dello spazio utilizzato

Frammentazione (2)

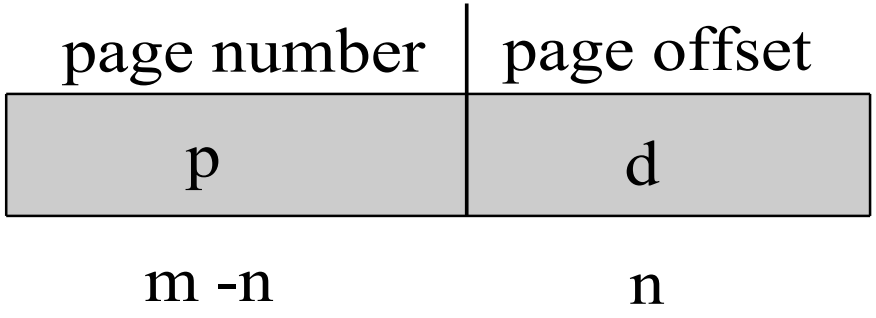
- Possibile rimedio: **compattazione**
  - Spostare le partizioni in maniera da poter unire buchi separati
  - Richiede binding in fase di esecuzione (rilocazione dinamica)
  - È molto onerosa in termini computazionali
  - Inoltre se il processo effettua I/O non può essere rilocato; alternativamente, l'I/O va fatto solo in buffer interni al sistema operativo
- La frammentazione è un problema generale che si verifica anche nelle memorie secondarie

Paginazione

- Idea: permettere una allocazione di memoria ai processi *noncontigua*
- La memoria fisica viene divisa in **frames**, ossia blocchi di dimensione fissa (tra 512 bytes e 16 Mbytes)
- Similmente lo spazio di indirizzi logico è diviso in **pagine**, ossia blocchi delle stesse dimensioni dei frames
- Una **tabella delle pagine** associa le pagine ai frames, e permette alla MMU di tradurre gli indirizzi logici in fisici
- Vantaggi:
  - Non vi è più frammentazione esterna
  - Lo spazio degli indirizzi logici è separato da quello degli indirizzi fisici (ad esempio, si possono avere indirizzi logici a 64 bit anche se la memoria fisica è più piccola)
- Svantaggi: frammentazione interna

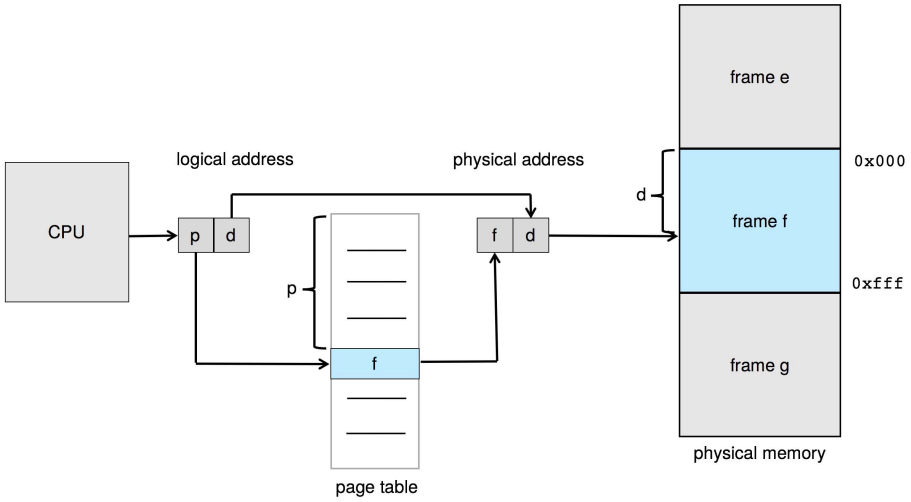
Paginazione: traduzione degli indirizzi logici (1)

- Un indirizzo logico è diviso in:
  - Numero di pagina (p): usato come indice nella tabella delle pagine
  - Offset di pagina (d): offset all'interno della pagina (identico all'offset nel frame)
- Se lo spazio di indirizzi logici ha dimensione 2^m…
- …e le pagine dimensione 2^n…
- …il numero totale di pagine è 2^{m-n}



Paginazione: traduzione degli indirizzi logici (2)

- La MMU traduce un indirizzo logico in fisico in questo modo:
  - Estrae il numero di pagina dall'indirizzo logico
  - Utilizza il numero di pagina per ricavare dalla tabella delle pagine il corrispondente numero di frame
  - Concatena il numero di frame all'offset di pagina e ottiene l'indirizzo fisico
- Vedere il libro per diversi esempi



Frammentazione interna e dimensione di pagina

- Nel caso medio la frammentazione interna è di mezzo frame per processo
- Questo suggerisce di fare pagine di dimensioni ridotte per ridurre lo spreco di memoria
- Ma in tal caso aumenta il numero totale di pagine, e di conseguenza la dimensione della tabella delle pagine
- Per tale motivo nel tempo la tendenza è stata di avere pagine di dimensioni maggiori: ad esempio, Windows 10 supporta pagine di 4 Kbyte e pagine di 2 Mbyte

Supporto alla paginazione nel sistema operativo

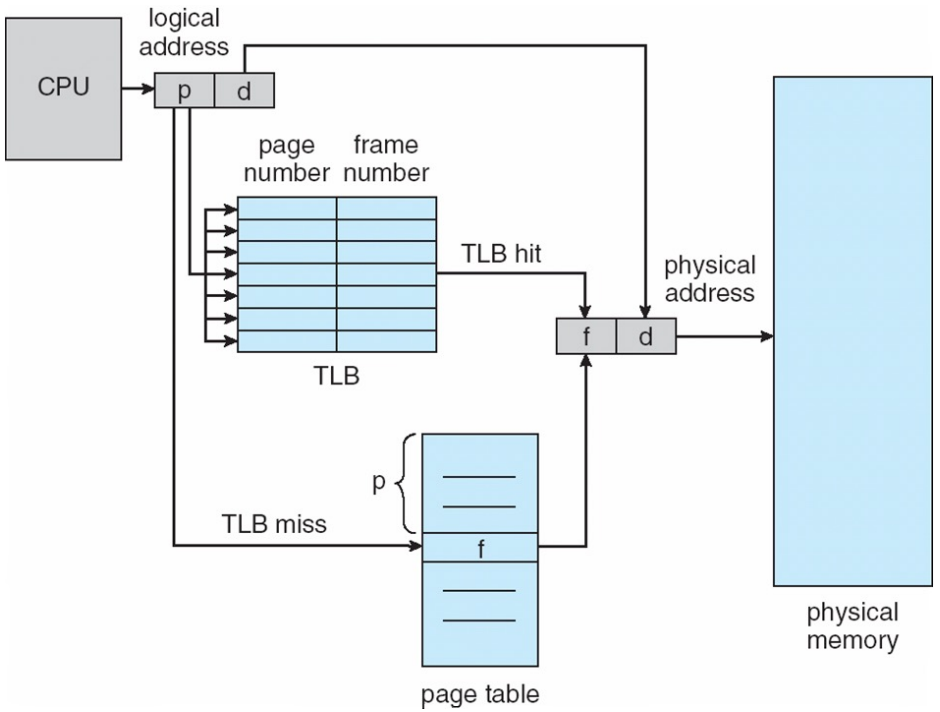
- Il sistema operativo deve mantenere la tabella delle pagine di ciascun processo
- Il sistema operativo deve mantenere una **tabella dei frame**, che indica lo stato di ogni frame (libero o assegnato, e in tal caso a che processo/i)
- Ricordiamo inoltre che il sistema operativo, se deve accedere alla memoria di un processo, deve effettuare la traduzione degli indirizzi logici del processo in indirizzi fisici (non può avvalersi della MMU)

Supporto hardware alla paginazione

- La tabella delle pagine è mantenuta in memoria centrale (è di regola troppo grande per avere dei registri ad hoc)
- La MMU utilizza due registri:
  - **Page table base register** (**PTBR**): indirizzo fisico dell'inizio della tabella delle pagine
  - **Page table length register** (**PTLR**): dimensione della tabella delle pagine
- In tale schema ogni accesso a dati/istruzioni richiede due accessi in memoria, uno alla tabella delle pagine, ed uno per recuperare il dato
- Essendo ciò troppo oneroso, si fa ricorso ad una cache apposita per la tabella delle pagine chiamata **translation lookaside buffer** (**TLB**)

Translation lookaside buffer

- Il TLB è di solito piccolo (da 64 a 1024 entries)
- Principale problema: ad ogni cambio di contesto devo effettuare il flush del TLB, il che comporta una notevole riduzione di prestazioni
- Per tale motivo alcuni TLB memorizzano un **address space identifier** (**ASID**) nelle loro entry, che identificano univocamente uno spazio di indirizzi, così da poter mantenere le entry di più tabelle delle pagine
- Gli ASID sono anche usati per implementare la protezione della memoria



Tempo effettivo di accesso alla memoria

- Ho un **TLB hit** se il numero di pagina di un indirizzo logico si trova nel TLB, altrimenti ho un **TLB miss**
- Il **tasso di successi** (**hit ratio**) è dato dalla percentuale di TLB hit sul totale degli accessi
- Supponendo che il tempo di accesso alla memoria sia *ma* e lo hit ratio sia *hr*, il **tempo medio di accesso alla memoria**, o **effective access time** (**EAT**) è:

$$EAT = ma\ hr + 2\ ma\ (1 - hr) = ma\ (2 - hr)$$

- (non teniamo conto delle maggiori o minori probabilità di cache hit o miss)

Politiche di sostituzione nel TLB

- Se capita un TLB miss ma il TLB è pieno, occorre sostituire un entry
- Diverse strategie:
  - Last recently used (LRU)
  - Round-robin
  - Casuale
- Alcuni processori generano un interrupt in maniera da permettere al sistema operativo di partecipare alla decisione di quale entry sostituire
- Alcuni processori permettono al sistema operativo di vincolare (wire down) delle TLB entry che non vogliono siano mai sostituite

Paginazione: protezione della memoria

- Realizzati con bit di protezione contenuti nella tabella delle pagine, che permettono di indicare se il frame associato è:
  - Read-only o read-write
  - Eseguibile o no
  - Combinazioni possibili
- Il registro PLTR permette di «tagliare» la tabella delle pagine in maniera da ridurre gli indirizzi logici disponibili al processo
- Alternativa: mettere in ogni entry della tabella delle pagine un bit di validità, che indica se l'entry è valida (la pagina è nello spazio di indirizzi logici) o no
- Vantaggi bit validità: maggiore flessibilità in quanto posso creare intervalli di indirizzi logici inaccessibili in qualsiasi punto dello spazio di indirizzi logici
- Vantaggi registro PLTR: si può ridurre la dimensione della tabella delle pagine

Pagine condivise

- Con la paginazione è facile condividere memoria fisica tra più processi: è sufficiente che i frame siano nelle tabelle delle pagine dei processi che condividono
- Applicazioni:
  - Condividere il codice (in maniera read-only!) tra più processi, ad esempio il codice delle librerie di sistema, risparmiando così spazio in memoria
  - Realizzare comunicazione interprocesso tramite memoria condivisa
- Attenzione: il codice condiviso deve essere rientrante!

Struttura delle tabelle delle pagine

- Con spazi di indirizzamento logici grandi (32 o 64 bit) le tabelle delle pagine possono diventare a loro volta molto grandi:
  - Abbiamo detto che, se le pagine sono grandi 2^n e lo spazio di indirizzamento logico 2^m, le pagine sono in tutto 2^{m-n}
  - Supponendo che ogni entry occupa e bytes di dimensione, in totale la tabella delle pagine ha dimensione $e \cdot 2^{m-n}$
  - Esempio: in un processore a 32 bit con pagine di 4 KB (12 bit) ed entries di 4 bytes, la dimensione della tabella delle pagine è 4 MB
  - In un processore a 64 bit, a parità degli altri parametri, la tabella delle pagine dovrebbe avere dimensione di 16 PB!
- Per tale motivo esistono varie soluzioni per strutturare le tabelle delle pagine in maniera che siano sufficientemente piccole/efficienti:
  - Tabelle delle pagine gerarchiche
  - Tabelle delle pagine di tipo hash
  - Tabelle delle pagine invertite

Tabelle delle pagine gerarchiche

- Servono per evitare di allocare una tabella delle pagine in una regione di memoria contigua
- Idea: paginare la tabella delle pagine
- Tabella a due livelli:
  - Ogni numero di pagina è diviso a sua volta in un numero di pagina e in un offset
  - Questi sono utilizzati per recuperare da una tabella esterna delle pagine l'indirizzo della pagina della tabella delle pagine
  - Questa, infine, è utilizzata per costruire l'indirizzo fisico
- Schemi a più di due livelli sono possibili

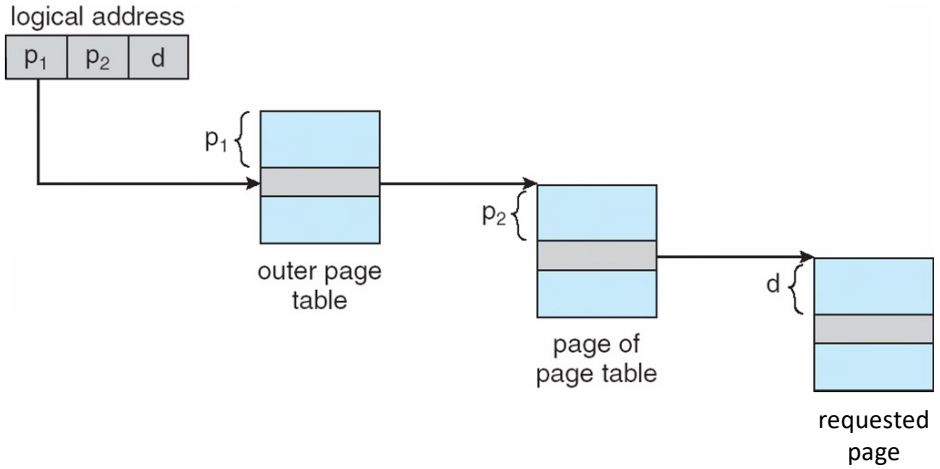


Tabelle delle pagine gerarchiche: vantaggi e svantaggi

- Un vantaggio è che la tabella delle pagine gerarchica permette di supportare contemporaneamente pagine di dimensioni diverse
  - Basta marcare un'entry nella tabella delle pagine esterne perché sia considerata un'entry di ultimo livello
  - Soluzione usata ad esempio nelle architetture IA-32 (a destra) e ARMv8
  - Permette di ridurre il numero di livelli (e quindi accessi in memoria dopo un TLB miss) e la dimensione della tabella
- Lo svantaggio principale è che aumenta il numero di accessi in memoria per recuperare un dato/istruzione nel caso pessimo
- Per tale motivo con i processori a 64 bit si tende a non usarle

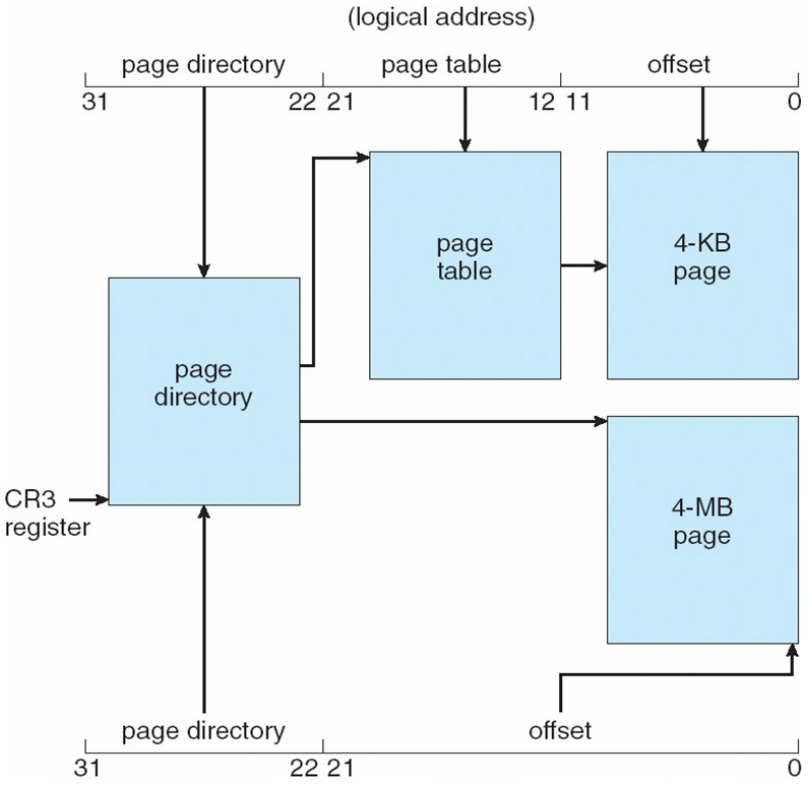


Tabelle delle pagine di tipo hash

- Idea: la tabella delle pagine è organizzata come una tabella hash
- Si applica una funzione hash (semplice!) al numero di pagina
- Ogni entry della tabella ha una lista concatenata di elementi per gestire le eventuali collisioni

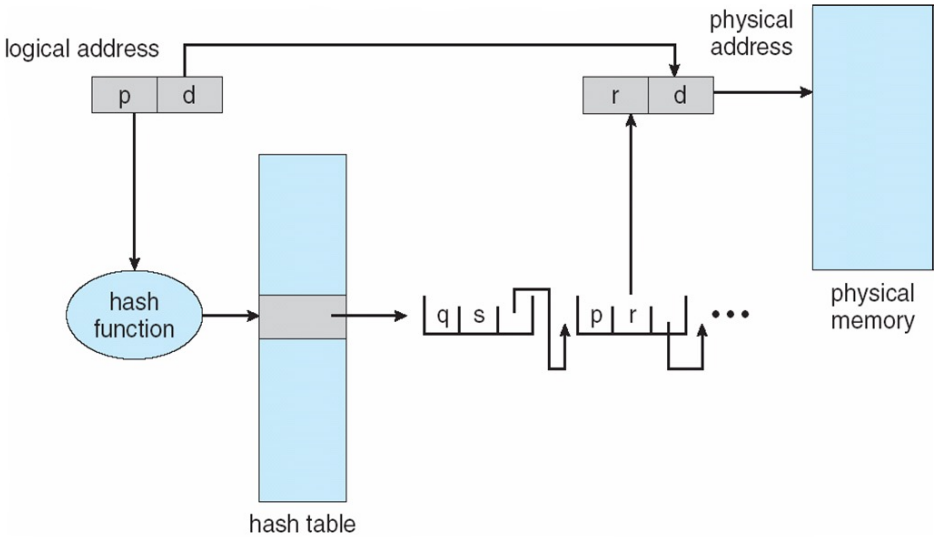
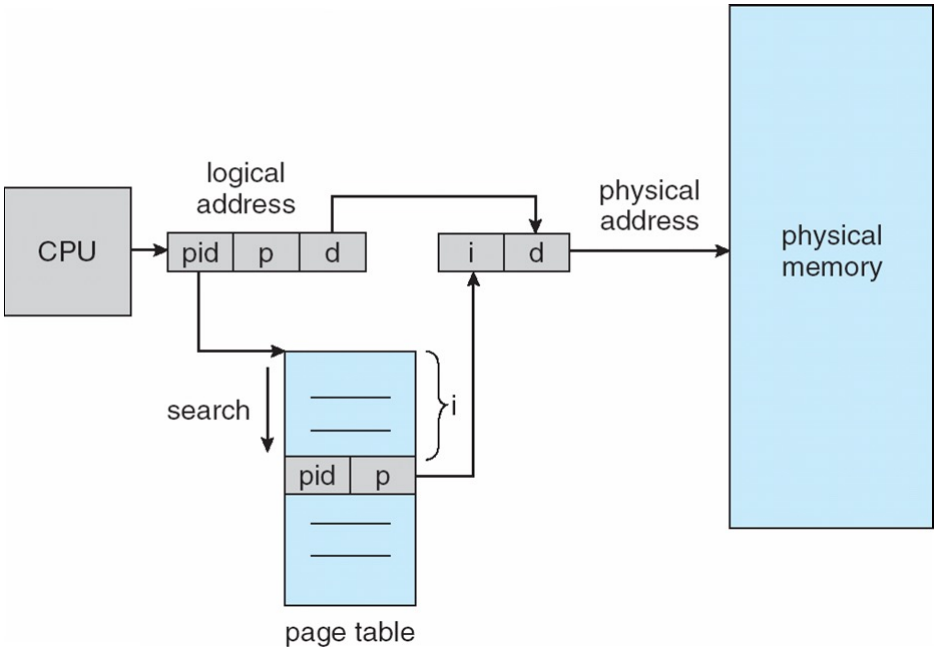


Tabelle delle pagine invertite

- Idea: un entry per ogni frame, anziché per ogni pagina (vengono tracciati i frame anziché le pagine)
- Ogni entry riporta il numero di pagina del corrispondente frame, più informazioni addizionali tra cui L'ASID (necessario!)
- Vantaggi:
  - Una sola tabella per tutti i processi
  - Se lo spazio di indirizzi fisici è più piccolo dello spazio di indirizzi virtuali, c'è un ulteriore risparmio
- Svantaggi:
  - Maggiore tempo di accesso (occorre fare una ricerca per trovare l'entry); mitigabile con tabella hash e TLB
  - Non è possibile condividere pagine tra processi diversi
- Utilizzata nelle architetture Power e UltraSPARC 64

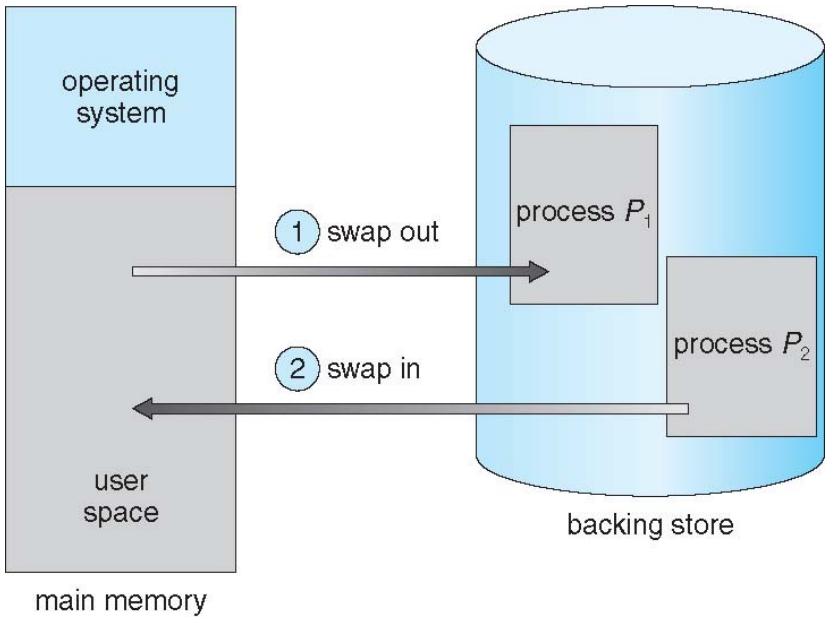


Swapping

- Lo swapping è una tecnica che permette di eseguire più processi di quanti la memoria fisica ne possa contenere
- Idea: spostare temporaneamente un processo pronto o in attesa dalla memoria centrale in una memoria secondaria (backing store, di solito il disco)
- L'obiettivo è permettere ad un altro processo di andare in memoria, e quindi in prospettiva di andare in esecuzione
- Possibili varianti:
  - Swapping standard (quello che di solito si intende con «swapping»)
  - Swapping con paginazione (quello che di solito si intende con «paginazione»)

Swapping standard

- Un intero processo viene spostato dalla memoria centrale alla backing store (**swap out**)
- Occorre spostare anche tutte le strutture dati del sistema operativo relative al processo e ai threads
- Svantaggio: spostare un intero processo è molto oneroso (solo Solaris usa ancora lo swapping standard in circostanze estreme)



Swapping con paginazione

- Sposta solo un sottoinsieme delle pagine di un processo, fino a quando non vi è sufficiente memoria per caricare l'immagine del nuovo processo
- Molto meno oneroso dello swapping standard
- L'operazione di scaricamento di una pagina dalla memoria centrale è detta **page out**, l'operazione inversa è detta **page in**