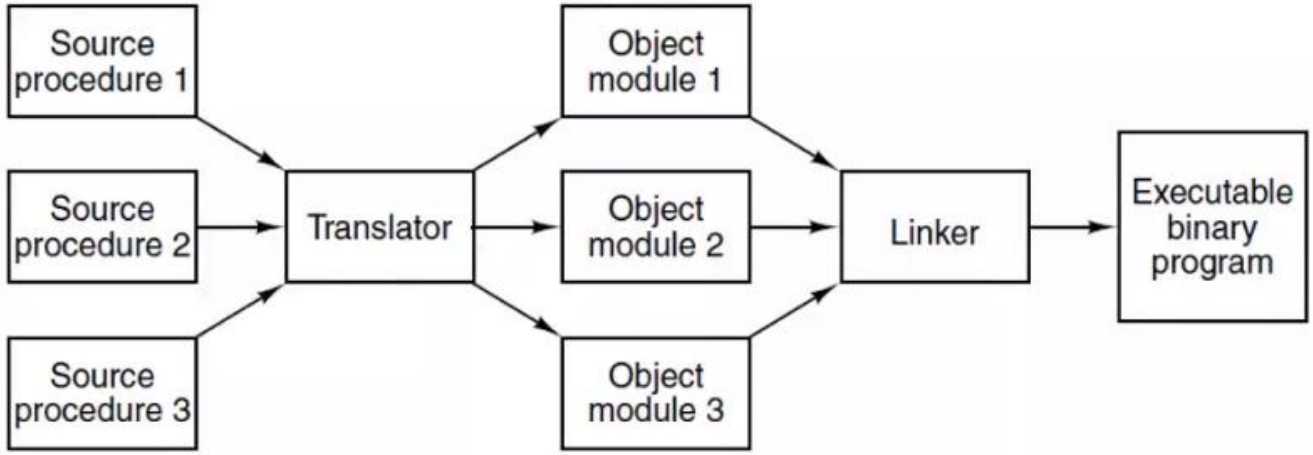
# Librerie condivise

- La condivisione può essere fatta con granularità diversa dalle singole pagine.
- Se un programma viene avviato due volte, la maggior parte dei sistemi operativi condividono automaticamente tutte le pagine di testo in modo che solo una copia è in memoria.
- Le pagine di testo sono sempre di sola lettura, quindi non c'è nessun problema.
- A seconda del sistema operativo, ciascun processo può avere la propria copia privata delle pagine dei dati o possono essere condivise contrassegnandole come **read-only**.
- Se un processo modifica una pagina di dati, gli viene fatta una copia privata (**copia on write**).

# Collegamento statico a libreria




- Dopo la fase di traduzione, il **linker** esegue il collegamento dei programmi oggetto. In essi qualsiasi **riferimento esterno non definito** (chiamata a funzione di sistema come **printf()**) è ricercato e incluso dalle librerie.
- Al termine del collegamento il **linker** scrive un programma binario eseguibile che contiene tutte le funzioni (dalle librerie sono incluse solo le quelle necessarie).

# Librerie condivise

- Se un programma utilizza pesantemente i moduli dell'interfaccia grafica potrebbe arrivare ad includere anche 50 MB di funzioni!
- Una diversa tecnica è di utilizzare le librerie condivise (**DLL** o **Dynamic Link Libraries** in Windows).
- Quando un programma è collegato con le librerie condivise invece di includere la funzione richiamata il **linker** include una piccola routine di **stub** che permette di attivare la funzione in fase di esecuzione (**run-time**).
- In questo modo se un altro programma ha già caricato la libreria condivisa, non vi è alcuna necessità di ricaricarla.
- Si noti che quando una libreria condivisa viene utilizzata non viene letta in memoria l'intera libreria.

# Librerie condivise

- Oltre a rendere i file eseguibili più piccoli e risparmiare spazio in memoria, le **librerie condivise** hanno un altro vantaggio:
  - Se una di esse viene aggiornata (es. correzione di **bug**), non è necessario ri-compilare i programmi che la utilizzavano perché i vecchi file binari continuano a funzionare.
- Svantaggio: non consentono la **rilocazione del codice al volo**, necessaria quando diversi processi accedono alla stessa libreria, due possibili soluzioni
  - **copy-on-write** delle pagine per ciascun processo (ma così si perdono i vantaggi della condivisione).
  - compilare le librerie con uno speciale flag che permette al compilatore di non produrre le istruzioni che utilizzino indirizzi assoluti (**codice indipendente dalla posizione**).

# Memory-mapped file

- Le librerie condivise sono un caso particolare di una struttura più generale chiamata file mappati in memoria o **memory-mapped file**.
- Un processo può inviare una chiamata di sistema per mappare un file in una porzione del suo spazio di indirizzi virtuali.
- I file mappati in memoria forniscono un modello alternativo per l'I/O: invece di fare letture/scritture, si accede al file come un grande array di caratteri in memoria.
- Se due o più processi mappano lo stesso file contemporaneamente, essi possono comunicare attraverso la memoria condivisa.
- Quando il SO dispone dei file mappati in memoria, si possono utilizzare per la gestione delle librerie condivise.

# La pulizia delle pagine

- Il paging funziona meglio quando ci sono molti frame liberi che possono essere utilizzati quando si verifica un **page-fault**.
- Se tutti i frame sono allocati con pagine modificate, prima di poter caricare in memoria una nuova pagina è necessario scrivere su disco quella sostituita.
- Per garantire l'esistenza di frame liberi, molti sistemi di paging hanno un processo background chiamato il **paging daemon** (**demone di paginazione**) che ispeziona periodicamente lo stato della memoria:
  - Se ci sono pochi frame liberi, inizia la selezione delle pagine da rimuovere utilizzando un algoritmo di sostituzione delle pagine.

# Politica di ripulitura

- La politicha di ripulitura può utilizzare l'algortimo dell'orologio a due passate:
  - Primo giro: se la pagina è sporca è scritta sul disco, altrimenti va avanti.
  - Secondo giro: si usa l'algoritmo standard di sostituzione delle pagine aumentando così la probabilità di trovare una pagina pulita.

# Interfaccia della memoria virtuale

- La memoria virtuale è trasparente ai processi e ai programmatori: essi vedono un grande spazio di indirizzamento virtuale su un computer con memoria fisica inferiore.

- In alcuni sistemi avanzati, i programmatori hanno un certo controllo sulla mappa di memoria e la possono utilizzare in modi non tradizionali per migliorare il comportamento del programma.

- Uno dei motivi per dare il controllo ai programmatori sulla loro mappa della memoria è quello di permettere due o più processi di condividere la stessa memoria.

# Memoria condivisa distribuita

- Un'altra tecnica di gestione avanzata della memoria è chiamata **distribuited shared memory** (o **memoria condivisa distribuita**).

- L'idea di base è quella di consentire a processi in rete di condividere un insieme di pagine come uno spazio di indirizzamento **lineare** e **condiviso**.

- Quando un processo fa riferimento a una pagina che non possiede genera un **page-fault**:
  - Il gestore dell'errore di pagina localizza la macchina che possiede la pagina e le invia un messaggio di richiesta di rimozione della pagina dalla mappa e inviarla nella rete.
  - Quando la pagina arriva, essa è mappata e l'istruzione che ha provocato l'errore viene riavviata.

# ASPETTI REALIZZATIVI

- Ci sono quattro momenti in cui il sistema operativo è coinvolto con paginazione:
  1) Durante la creazione del processo.
  2) Durante l'esecuzione del processo.
  3) Durante un **page-fault**.
  4) Durante la chiusura del processo.

# Creazione del processo

- Quando viene creato un nuovo processo, il sistema operativo deve:
  1) Determinare quante pagine assegnare al programma e ai dati e creare le corrispondenti tabelle delle pagine.
  2) Allocare e inizializzare in memoria lo spazio per la tabella pagine.
  3) Riservare uno spazio per garantire lo swap su disco delle pagine.
  4) Inizializzare l'area di swap su disco con programma e dati così che quando il nuovo processo parte i page-fault causeranno il caricamento in memoria delle pagine.
  5) Registrare nella tabella dei processi tutte le informazioni della tabella pagine e dell'area di swap su disco.

# Esecuzione del processo

- Quando un processo è schedulato per l'esecuzione, la MMU deve essere ripristinata per il nuovo processo e la TLB deve essere vuotata (nessuna informazione del processo precedentemente in esecuzione dovrà essere mantenuta).

- La nuova tabella delle pagine deve diventare quella corrente, di solito copiandola o copiando un puntatore ad essa in un dato registro hardware.

# Page fault

- Quando si verifica un errore di pagina, il sistema operativo deve:
  1) leggere i registri hardware per determinare quale indirizzo virtuale ha causato il **page-fault**.
  2) Calcolare la pagina mancante localizzandola sul disco.
  3) Trovare un frame disponibile per collocare la nuova pagina, rimuovendo la vecchia.
  4) Leggere la pagina dal disco.
  5) Ripristinare il Program Counter per far in modo che punti all'istruzione che ha provocato l'errore in modo da ri-eseguirla.

# Process termination

- Quando un processo termina, il sistema operativo deve rilasciare la sua tabella delle pagine, le pagine e lo spazio su disco che le pagine occupano.

- Se alcune delle pagine sono condivise con altri processi, le pagine in memoria e su disco possono essere rilasciate solo quando l'ultimo processo che le utilizza è terminato.

# Gestione dei page fault

- Quando accade un **page-fault** si verificano i seguenti eventi:
  1) L'hardware esegue una trap nel kernel, salvando il PC nello stack.
  2) Viene avviata una routine in codice assembly per salvare i registri generali e le altre informazioni volatili.
  3) Il sistema operativo cerca di scoprire quale pagina virtuale è richiesta (potrebbe essere scritto in un registro hardware).
  4) Una volta che l'indirizzo virtuale che ha causato l'errore è noto, il sistema operativo verifica se:
     - l'indirizzo è valido e la protezione coerente con l'accesso,
     - c'è un frame libero lo usa altrimenti attiva l'algoritmo di sostituzione delle pagine che trova una pagina vittima.
  5) Se la pagina selezionata è sporca, viene schedulata per il trasferimento sul disco e ha luogo un cambio di contesto: viene sospeso il processo in **page-fault** ed eseguito un altro processo fino a quando il trasferimento del disco non è completato.

# Gestione dei page fault

6) Non appena il frame è stato scritto sul disco (ed è pulito), il sistema operativo schedula un'operazione per portare la pagina in memoria. Mentre la pagina viene caricata, il processo che ha provocato l'errore è ancora sospeso e un altro processo utente è eseguito.
7) Quando l'interrupt del disco indica che la pagina è arrivata, le tabelle delle pagine vengono aggiornate (il frame è contrassegnato con lo stato normale).
8) L'istruzione che ha provocato l'errore viene ripristinata e il PC è riportato all'istruzione che ha generato l'errore.
9) Il processo in errore è schedulato e il sistema operativo ritorna alla routine (in linguaggio assembly) che lo ha chiamato.
10) Questa routine ricarica i registri e altre informazioni di stato, quindi ritorna allo spazio utente per proseguire l'esecuzione, come se non ci fosse stato il **page-fault**.

# Backup delle istruzioni

- Quando un programma fa riferimento a una pagina che non è nella memoria, l'istruzione che ha causato il **page-fault** viene interrotta e avviene una trap per il sistema operativo.
- Dopo che il sistema operativo ha recuperato la pagina mancante è necessario riavviare l'istruzione che causa la trap.
- Un registro interno nascosto in alcune CPU salva il PC prima di eseguire ogni istruzione:
  - Queste macchine possono anche avere un secondo registro che indica quali registri sono già stati autoincrementati o autodecrementati e di quanto.
- Il sistema operativo utilizza queste informazioni al fine di ripristinare lo stato precedente prima dell'istruzione ha provocato l'errore e poterla riavviare come se la pagina fosse stata in memoria.
- Se l'hardware non dispone di questi registri è il sistema operativo che deve capire cosa è successo e come ripristinare l'istruzione in errore.

# Blocco delle pagine in memoria

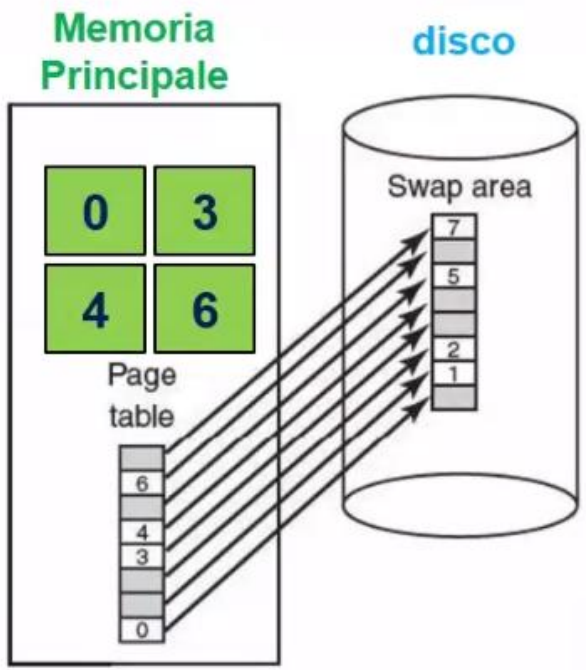
- Memoria virtuale e I/O interagiscono in modo sottile.
- Si consideri un processo che vuole leggere un file all'interno del proprio spazio di indirizzamento, mentre è in attesa dell'I/O viene sospeso e un altro processo va in esecuzione.
  - Quest'ultimo ottiene un errore di pagina.
- Se l'algoritmo di paginazione è globale, c'è una piccola probabilità (non zero) che la pagina contenente il buffer di I/O venga rimossa dalla memoria.
- Se il dispositivo di I/O è pronto per eseguire il trasferimento DMA a quella pagina, rimuovendolo i dati finora scritti finirebbero di nuovo sul disco. Esistono 2 soluzioni:
  - Bloccare le pagine in memoria che stanno effettuando un I/O in modo che non vengano rimosse (**pinning nella memoria**).
  - Eseguire tutte le operazioni di I/O all'interno dei buffer del kernel e successivamente copiare i dati su pagine utente.

# Swapping dei processi su disco

- Quando una pagina viene rimossa dalla memoria non sappiamo dove è memorizzato sul disco.
- La tecnica semplice è di utilizzare una partizione dedicata (di swap) o, ancora meglio, su un disco separato (per meglio bilanciare il carico dell'I/O).
- La maggior parte dei sistemi UNIX funzionano in questo modo.
- Quando il sistema viene avviato, questa partizione di swap è vuota ed è rappresentato in memoria come una singola voce, con un inizio e la dimensione.
- Nello **swapping statico** ad ogni processo corrisponde sul disco uno spazio identico a quello occupato in memoria. Quando i processi terminano il loro spazio su disco viene liberato.
- Nella tabella dei processi è memorizzato soltanto l'indirizzo di inizio sul disco della sua area di swap.

# Paging su un'area di swap statica

- Le pagine 0, 3, 4, e 6 sono in memoria, mentre le pagine 1, 2, 5 e 7 sono sul disco.
- L'area di swap sul disco è grande quanto il processo (otto pagine), ogni pagina ha una posizione fissa che viene scritta quando la pagina è rimossa dalla memoria.
- Le pagine vengono memorizzate in modo continuo in funzione del loro numero di pagina virtuale.
- Tutte le pagine in memoria hanno una copia **ombra** (o **shadow**) su disco, che potrebbe non essere aggiornata se la pagina è stata modificata.

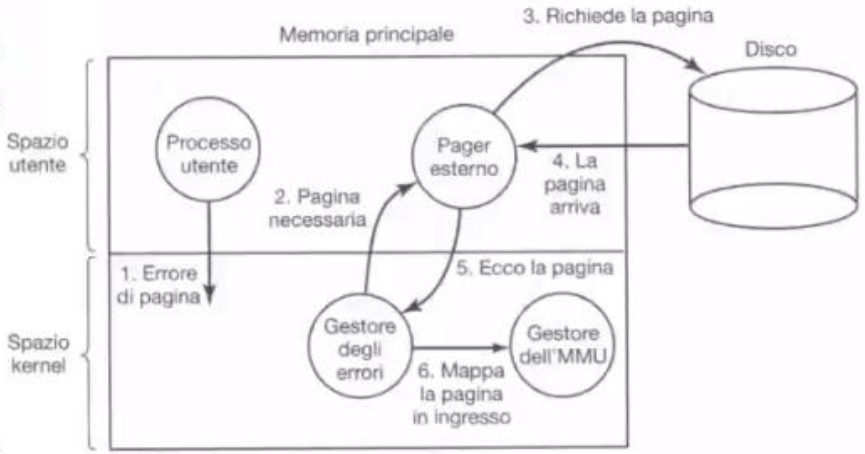


# Separazione delle politiche dal meccanismo

- Un importante strumento per gestire la complessità di un sistema è quello di separare la politica dal meccanismo.
- Questa teoria può essere applicata alla gestione della memoria in quanto il gestore della memoria è eseguito come processo a livello utente.
- Il sistema di gestione della memoria può essere diviso in tre parti:
  1) Un gestore a basso livello dell'MMU.
  2) Un gestore dei page-fault che è parte del kernel.
  3) Un **Pager** esterno in esecuzione nello spazio utente.

# Separazione delle politiche dal meccanismo

1) Un processo utente genera un page-falut.
2) Il gestore degli errori capisce quale pagina virtuale serve e attiva il Pager esterno.
3) Il Pager la richiede al disco.
4) Il Pager ottiene la pagina.
5) Il Pager la passa al gestore.
6) Il gestore chiede al gestore del MMU di mappare la pagina.
7) Il processo utente è riavviato.



# Separazione delle politiche dal meccanismo

- Dove si trova l'algoritmo di sostituzione delle pagine?
- Se sta nel Pager, esso non ha accesso a ai bit M e R di tutte le pagine fondamentali per qualsiasi algoritmo. Quindi occorre passargli queste informazioni.
- Se sta nel kernel si ottiene modularità del codice e maggiore flessibilità.
  - Lo svantaggio principale è il sovraccarico causato dal passaggio dalla modalità kernel a quella utente.

# SEGMENTAZIONE

- La memoria virtuale illustrata fin ora è monodimensionale perché gli indirizzi virtuali vanno da 0 a un indirizzo massimo.
- Avere più spazi di indirizzi virtuali può essere utile. Per esempio un compilatore utilizza molte tabelle a mano a mano che procede nella compilazione:

  1) Il testo sorgente.
  2) La tabella dei simboli, contenente i nomi e gli attributi delle variabili.
  3) La tabella contenente tutte le costanti utilizzate.
  4) L'albero di parsing, contenente l'analisi statica del programma.
  5) Lo stack utilizzato per chiamate di procedura all'interno del compilatore.

# SEGMENTAZIONE

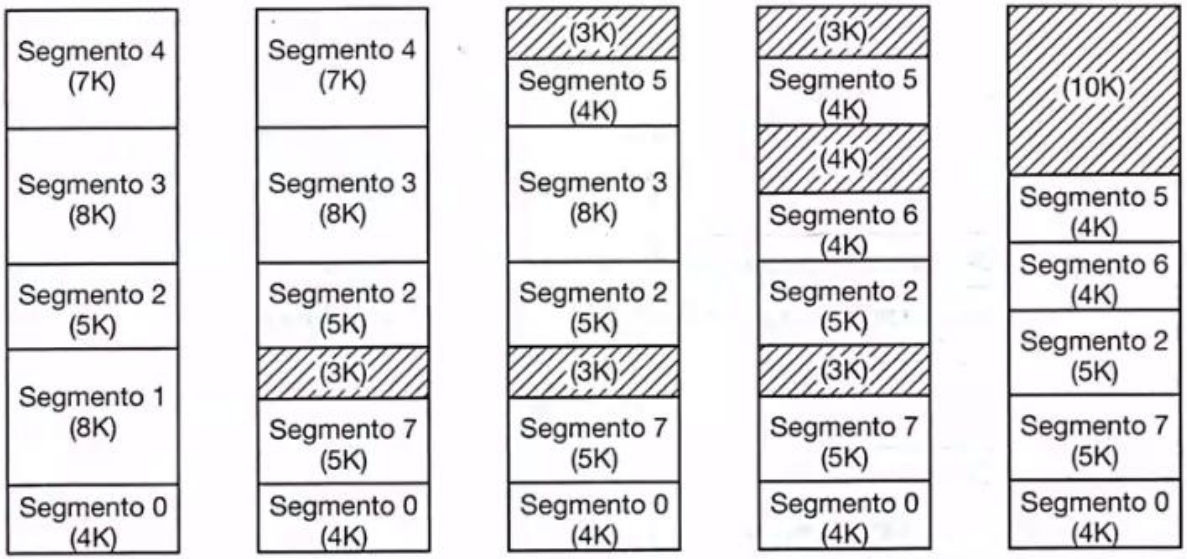
- Una soluzione generale è di fornire più spazi di indirizzi **indipendenti**, detti **segmenti**.
- Ogni segmento consiste in una sequenza lineare di indirizzi, da 0 a un valore massimo.
- Segmenti diversi possono avere lunghezze diverse che si modificano (+/-) durante l'esecuzione.
- In questo modello di memoria bidimensionale per specificare un **indirizzo**, sono necessari:
  1) Il numero del segmento.
  2) L'indirizzo all'interno del segmento.
- I segmenti sono entità logiche utilizzabili dal programmatore come accade per gli array, le procedure,...

# Vantaggi dei segmenti

- Si semplifica la gestione di strutture di dati che crescono o si riducono di dimensione.
- Se ogni procedura occupa un segmento separato (indirizzo iniziale 0) il **linking** delle procedure compilate separatamente è facile.
- La segmentazione semplificata la condivisione di procedure o dei dati tra diversi processi: le librerie condivise possono essere messe in un segmento e utilizzate tra più processi.
- I segmenti possono avere diversi tipi di protezione a seconda del loro contenuto (una procedura può essere eseguita e non scritta, un array utilizzato ma non eseguito).
- Per questa ragione normalmente è bene inserire in un segmento un solo tipo di oggetto.
- Mentre il programmatore sa cosa c'è dentro un segmento, il contenuto delle pagine e la loro gestione è totalmente invisibile al programmatore.

# Realizzazione della segmentazione pura

- La realizzazione della segmentazione differisce dalla paginazione perché la lunghezza delle pagine è fissa mentre quella dei segmenti varia.
- Dopo che il sistema è in funzione per un certo tempo, la memoria verrà suddivisa tra segmenti contenenti dati e zone di memoria libera (**frammentazione esterna**).



# Quadro di sintesi e confronto

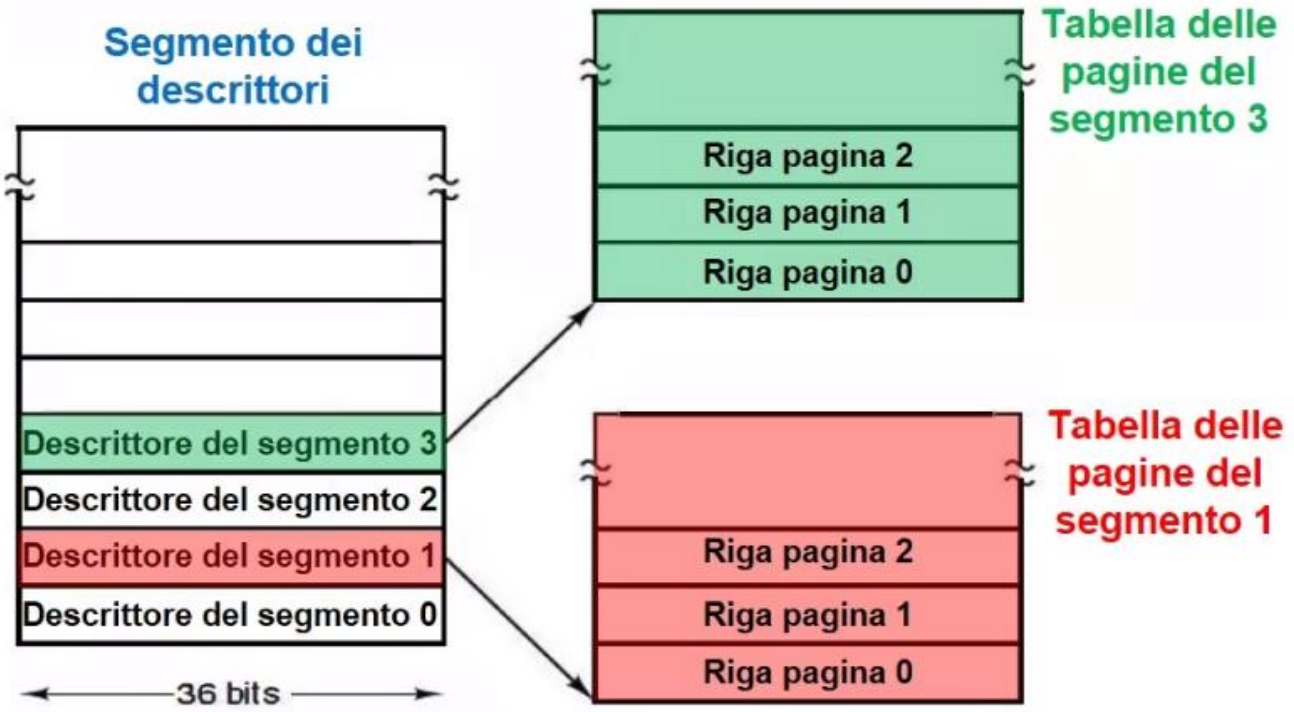
| Quesito | Paginazione | Segmentazione |
|---|---|---|
| **Questa tecnica è visibile al programmatore?** | No | Sì |
| **Quanti spazi lineari di indirizzo ci sono?** | 1 | molti |
| **lo spazio totale di indirizzamento può superare la dimensione della memoria fisica?** | Sì | Sì |
| **Le procedure e i dati possono essere separati e protetti?** | No | Sì |
| **Le tabelle la cui dimensione varia nel tempo possono essere ospitate facilmente?** | No | Sì |
| **La condivisione delle procedure tra gli utenti è agevolata?** | No | Sì |
| **Perché è stata inventata questa tecnica?** | Per ottenere un spazio di indirizzamento lineare più ampio senza acquistare memoria fisica | Per consentire ai programmi e ai dati di essere suddivisi in spazi di indirizzi logicamente indipendenti e per facilitarne la condivisione e la protezione |

# Segmentazione con paging: MULTICS

- Se i segmenti sono di grandi, non è conveniente mantenerli nella memoria principale.
- Una soluzione è quella di paginarli, in modo che siano in memoria solo le pagine che sono effettivamente necessarie.
- Molti sistemi hanno sostenuto la paginazione dei segmenti.
- MULTICS forniva a ciascun programma una memoria virtuale di al massimo di $\mathbf{2^{18}}$ segmenti (più di **250k**), ciascuno dei quali lungo al massimo **65.536** parole.
- Ciascun segmento è paginato così si hanno i vantaggi del paging con quelli della segmentazione.
- Ogni programma MULTICS ha una tabella dei segmenti, con un descrittore per segmento.
- Considerata la dimensione la tabella è segmentata e paginata.

# Memoria virtuale nel MULTICS

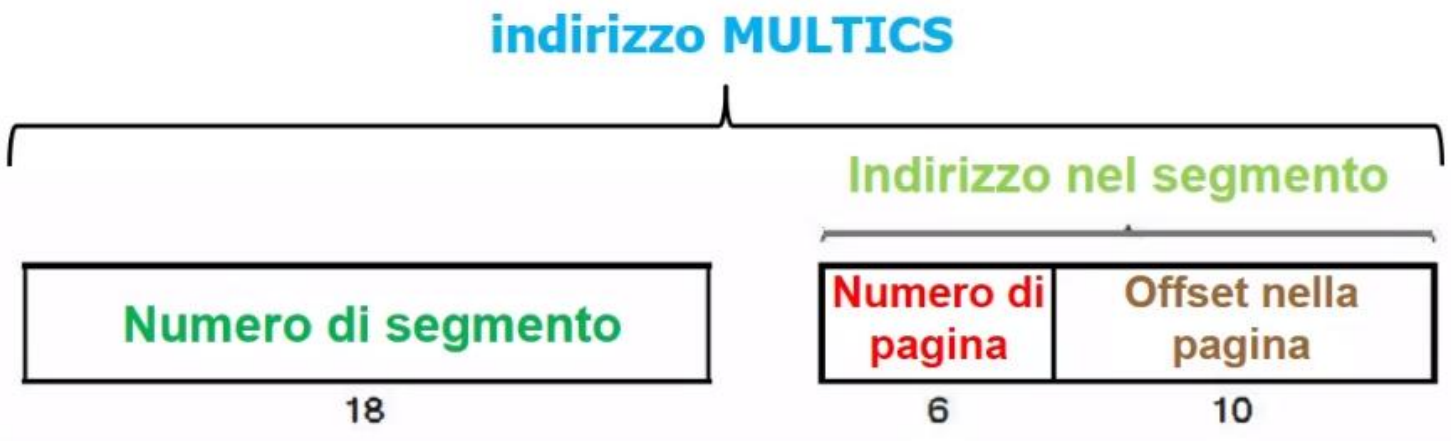
- Il descrittore del segmento contiene l'indicazione se il segmento è nella memoria principale o meno.
- Se una parte del segmento è in memoria allora esso è considerato in memoria ed anche la sua tabella delle pagine.



# Indirizzo virtuale nel MULTICS

- In MULTICS un **indirizzo** consiste di due parti: il **numero di segmento** e l'**indirizzo nel segmento**.

- L'indirizzo all'interno del segmento è ulteriormente suddiviso in un **numero di pagina** e un **offset** all'interno della pagina.



# Localizzazzione delle pagine nel MULTICS

- Per ogni riferimento di memoria:

1) Viene utilizzato il **numero di segmento** per trovare il descrittore del segmento.
2) La tabella pagine del segmento è in memoria?
   - Sì, lo localizza.
   - No, viene generato un **errore di segmento**.
3) Viene esaminata la riga della tabella delle pagine per la pagina virtuale: è in memoria?
   - Sì, Viene estratto l'indirizzo della pagina in memoria.
   - No, viene generato un **page-fault**.
4) Per localizzare la pagina è aggiunto l'**offset** all'inizio della pagina.
5) Legge o scrive la pagina.

# TLB del MULTICS

- Ripetere tutti questi passaggi ogni riferimento di memoria può rallentare l'intero sistema.
- MULTICS contiene una TLB alta velocità a 16 parole che può cercare, in un sol colpo, una chiave in tutte le sue voci.
- Ad ogni riferimento di memoria prima si controlla dentro la TLB: se la coppia <**segmento**, **indirizzo**> è presente troviamo direttamente il **frame** senza dover guardare nel segmento descrittore e nella tabella delle pagine.

| segmento | indirizzo | frame | protezione | età | riga usata? |
|---|---|---|---|---|---|
| 4 | 1 | 7 | Read/write | 13 | 1 |
| 6 | 0 | 2 | Read only | 10 | 1 |
| 12 | 3 | 1 | Read/write | 2 | 1 |
| | | | | | 0 |
| 2 | 1 | 0 | Execute only | 7 | 1 |
| 2 | 2 | 12 | Execute only | 9 | 1 |
| | | | | | |

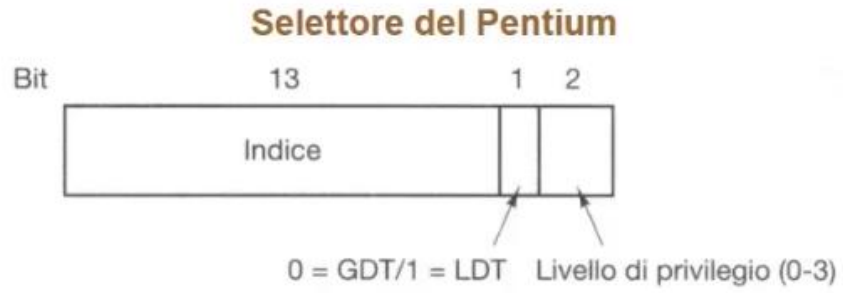
Versione semplificata di una **TLB** nel MULTICS

# Segmentazione con paging: Intel Pentium

- La memoria virtuale sul Pentium è simile a quella sul MULTICS:
- Mentre MULTICS ha **256K** segmenti indipendenti, ciascuno con al massimo **64K** di parole a **36** bit, il Pentium ha **16K** segmenti indipendenti, ciascuno dei quali fino a **1** miliardo di parole a **32** bit:
  - meno segmenti, più grandi (ciò che serve ai programmi).
- La memoria virtuale del Pentium è fondata su due tabelle:
  1) **LDT** (Local Descriptor Table).
  2) **GDT** (Global Descriptor Table).
- Ogni programma ha una propria tabella locale dei descrittori (**LDT**), ma vi è una singola tabella globale (**GDT**) condivisa da tutti i programmi sul computer.
- **LDT** descrive segmenti locali per ogni programma (codice, dati, stack,...) mentre il **GDT** descrive segmenti sistema (sistema operativo compreso).

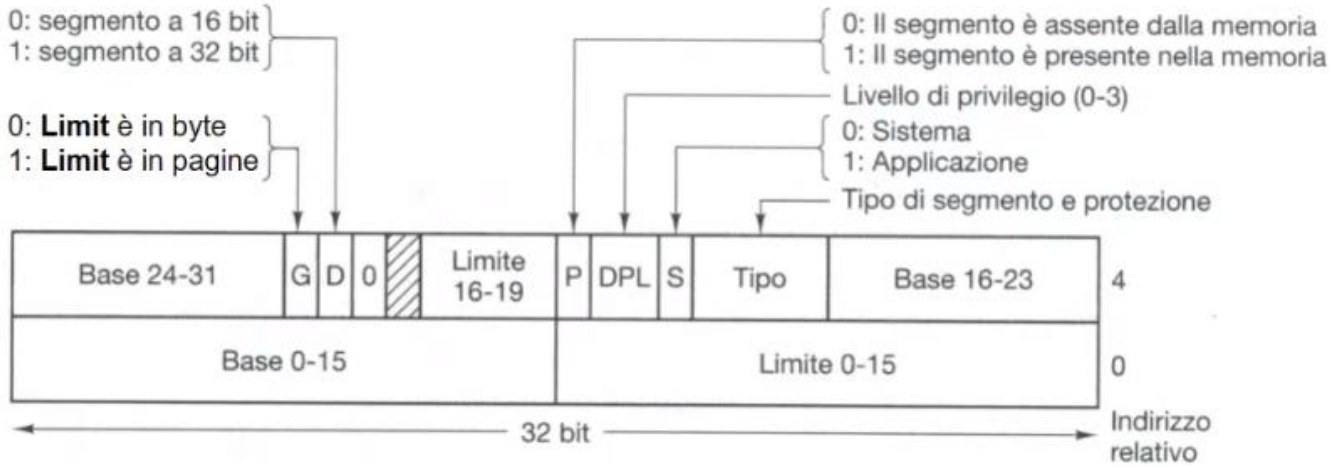
# Segmentazione con paging: Intel Pentium

- Per accedere ad un segmento, il programma prima carica un selettore per quel segmento in uno dei 6 registri segmento della CPU.
- Durante l'esecuzione, il registro **CS** contiene il selettore per il segmento di codice e il registro **DS** contiene il selettore per il segmento di dati.
- Ciascun selettore è un numero a 16 bit:
  1) 13 bit specificano la riga nella tabella dei descrittori.
  2) Un bit indica il tipo di tabella (0=**GDT**, 1=**LDT**).
  3) Due bit sono di protezione.



# Segmentazione con paging: Intel Pentium

- Quando un selettore viene caricato in un registro di segmento, il descrittore corrispondente è prelevato dalla **LDT** (o dalla **GDT**) e memorizzato nei registri del microprogramma.
- Un descrittore consiste di 4 byte, incluso l'indirizzo di base del segmento, la dimensione e altre informazioni.



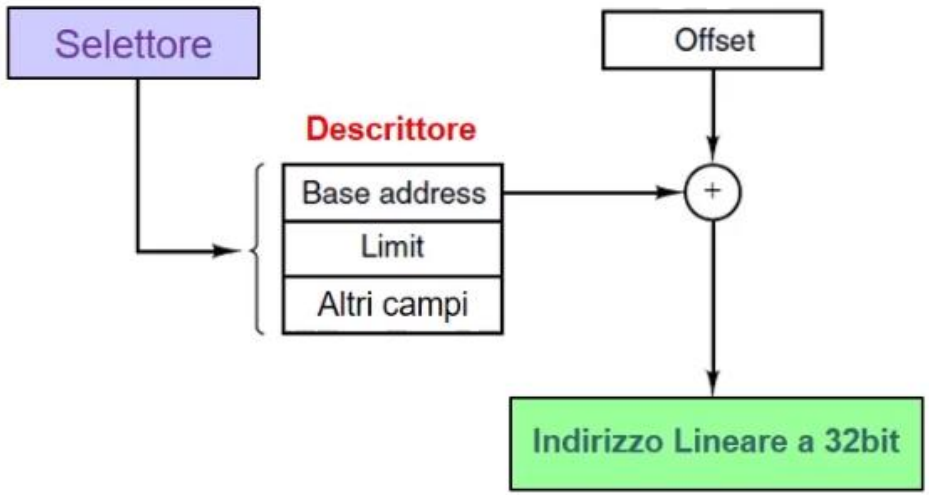

**Descrittore dei segmenti codice del Pentium**

# Mapping degli indirizzi del Pentium

- *Come convertire una coppia <selettore, offset> in un indirizzo fisico?*
- Appena il microprogramma sa quale registro di segmento è stato utilizzato, può trovare il descrittore completo nei suoi registri interni.
- Si verifica una **trap** se:
  - il segmento non esiste (selettore 0) o è salvato su disco per effetto del paging.
  - se l'offset è dopo la fine del segmento (**Limit**).
- Nel descrittore ci sono solo 20 bit disponibili (invece che 32).
- La dimensione del segmento è espressa in byte o pagine a seconda del valore del Granularity bit (**Gbit**):
  - 0, **Limit** è in byte (fino a 1 MB).
  - 1, **Limit** è in pagine.

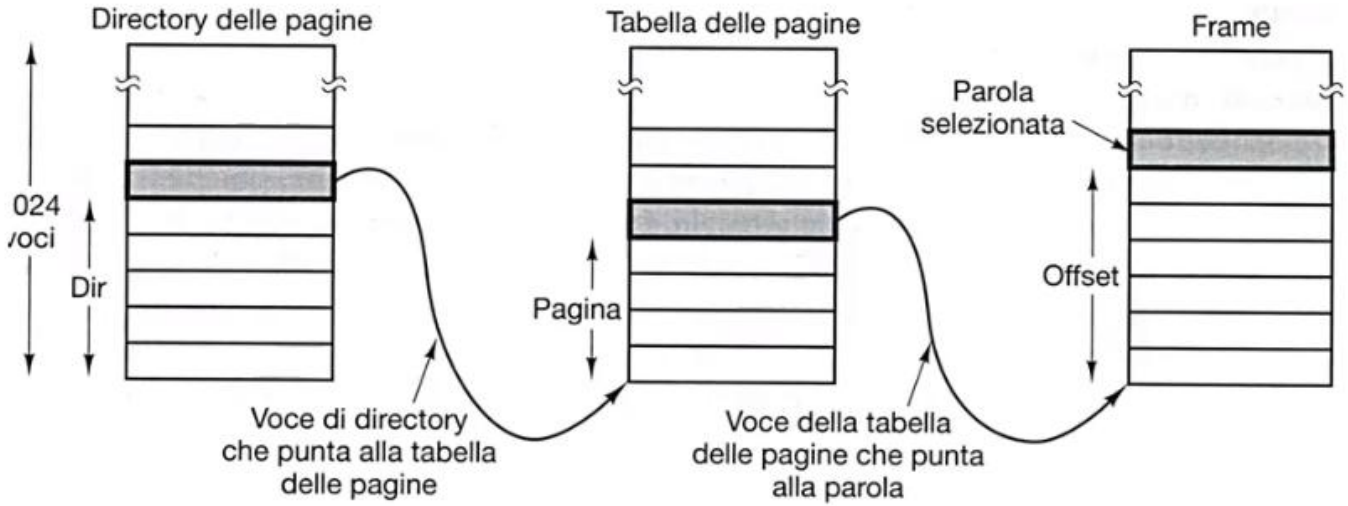
# Conversione in indirizzo lineare

- Poiché la dimensione della pagina nel Pentium è **4 KB**, di 32 bit a 20 sono sufficienti per il frame number.
- Se il segmento è in memoria e l'offset è nell'intervallo ammissibile, al campo Base (32 bit) del descrittore è aggiunto l'Offset per formare un **indirizzo lineare**.



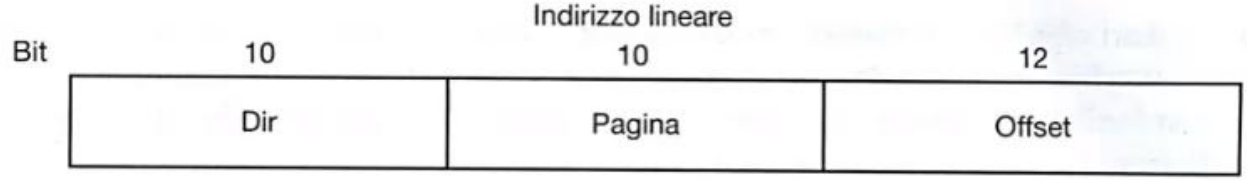
# Mapping dell'indirizzo linerare in fisico

- Ogni programma in esecuzione ha una **directory delle pagine** composta di **1024** voci a **32** bit.
- Un registro globale indica la riga di questa directory che punta a una **tabella delle pagine** contenente **1024** righe a **32** bit.
- Le voci della **tabella delle pagine** puntano a **frame di pagina**.



# Indirizzo lineare del Pentium

- L'**indirizzo lineare** è diviso in tre campi:




- Il campo **Dir** è l'indice nella **directory delle pagine**: individua un puntatore alla corretta **tabella delle pagine**.
- Il campo **Pagina** viene utilizzato come indice nella tabella delle pagine per trovare l'indirizzo fisico del **frame della pagina**.
- L'**Offset** viene aggiunto l'indirizzo del frame della pagina per ottenere l'indirizzo fisico del byte o della parola necessaria.

# Segmentazione con paging: Intel Pentium

- Le righe della **tabella delle pagine** sono lunghe **32** bit, 20 dei quali contengono il numero di **frame di pagina**.
- I restanti bit, impostati dall'hardware a beneficio del sistema operativo, sono:
  - bit di accesso,
  - dirty bit,
  - bit di protezione
  - altri bit di utilità,
- Ogni tabella delle pagine ha **1024** righe per frame di pagina a **4KB**, in modo che una singola tabella delle pagine gestisce **4MB** di memoria.
- Per evitare di fare accessi ripetuti alla memoria, il Pentium (come il MULTICS) ha una piccola TLB che mappa direttamente le combinazioni **<Dir, Pagina>** utilizzate più recentemente.

# Conclusioni

- Concluso gli argomenti inerenti le problematiche nella progettazione dei sistemi di paging.
- Introdotto gli elementi realizzativi nei sistemi di paging.
- Studiato la segmentazione e considerato alcuni casi reali (MULTICS e Pentium).