Struttura interna del sistema operativo Linux

3. La gestione della memoria virtuale

A cura di:
Anna Antola Giuseppe Pozzi

DEI, PoliMI, Milano

anna.antola/giuseppe.pozzi@polimi.it
- versione del 16 marzo 2003 -

Vincoli dei programmi

- Un processo occupa una quantità fissa di memoria.
- La memoria fisica contiene tutti i processi attivi.
- Un processo è caricato in celle consecutive di memoria.
- Due processi non possono condividere spazi di memoria.
- La dimensione di un processo è in ogni caso inferiore alla dimensione della memoria fisica.

Il concetto di memoria virtuale

- Separare il concetto di spazio di indirizzamento di un programma eseguibile e dimensione effettiva della memoria fisica
- Spazio di indirizzamento = numero di parole indirizzabili
  - Dipende esclusivamente dal numero di bit dell'indirizzo e non dal numero di parole di memoria effettivamente disponibile

Spazio di indirizzamento fisico e dimensioni della memoria fisica

- Dimensione della memoria fisica = n° di byte che la costituiscono
- Spazio di indirizzamento: definito dal numero di bit disponibili per specificare l'indirizzo di una cella di memoria
  - se N sono i bit di indirizzo, allora 2^N è il numero massimo di parole indirizzabili
- Poichè la memoria fisica deve essere tutta indirizzabile è sempre minore o uguale al suo spazio di indirizzamento

Codifica Esadecimale

HEX	Bin	HEX	Bin	HEX	Bin	HEX	Bin
0	0000	4	0100	8	1000	C	1100
1	0001	5	0101	9	1001	D	1101
2	0010	6	0110	A	1010	E	1110
3	0011	7	0111	B	1011	F	1111

- 32 bit di indirizzo (4Giga indirizzabili)
- 0000FFFF = 0000 0000 0000 0000 1111 1111 1111 1111

che equivale all'indirizzo 2^{16}-1

Memoria virtuale (1)

- Indirizzi a cui fa riferimento il programma in esecuzione: indirizzi virtuali
- Gli indirizzi virtuali sono quelli generati da linker (a partire dall'indirizzo 0): sono quindi indirizzi rilocabili
- Lo spazio di indirizzamento virtuale è quello definito dalla lunghezza degli indirizzi virtuali
- In un programma è possibile definire la dimensione (in termini di indirizzi virtuali):
  - dimensione iniziale: è quella calcolata dal linker dopo la generazione del codice eseguibile
  - dimensione durante l'esecuzione: può variare a causa della modifica dello *stack* per le chiamate a sottoprogrammi e a causa della modifica dello *heap* per la gestione delle strutture dati dinamiche

Memoria virtuale (2)

- Indirizzi di memoria centrale: indirizzi fisici
- E' presente un meccanismo di **traduzione automatico degli indirizzi** virtuali in indirizzi fisici
- Con la memoria virtuale, la CPU produce un *indirizzo virtuale*, che viene tradotto da una combinazione di elementi hardware e software in un *indirizzo fisico*, che può essere utilizzato per accedere alla memoria principale: *traduzione dell'indirizzo* (*memory mapping*)

Memoria virtuale e rilocazione dinamica

- Avere degli indirizzi virtuali implica che ogni programma eseguibile generato dal collegatore sia in formato rilocabile, cioè con indirizzi che partono da 0
- Inoltre, al momento dell'esecuzione il programma viene caricato (opportunamente) in memoria con gli indirizzi virtuali
- Il meccanismo di **traduzione automatico degli indirizzi** virtuali in indirizzi fisici, è **attivato durante l'esecuzione** del programma ad ogni riferimento a memoria ed è trasparente al programmatore, al compilatore e al linker
- Questa traduzione consente il caricamento del programma in una **qualsiasi locazione** della memoria fisica

Obiettivi della memoria virtuale

- La dimensione di memoria di un programma può essere maggiore della dimensione della memoria fisica.
  - Il programmatore può scrivere i propri programmi pensando di avere a disposizione l'intero spazio di indirizzamento virtuale indipendentemente dalle dimensioni effettive della memoria centrale
- Questo implica che un programma in esecuzione non risiede completamente in memoria. Il meccanismo di caricamento in memoria delle parti del programma e la traduzione degli indirizzi è trasparente al programmatore
- Più programmi in esecuzione (processi) possono risiedere contemporaneamente in memoria, sfruttandola in modo efficiente

LINUX e la memoria virtuale (1)

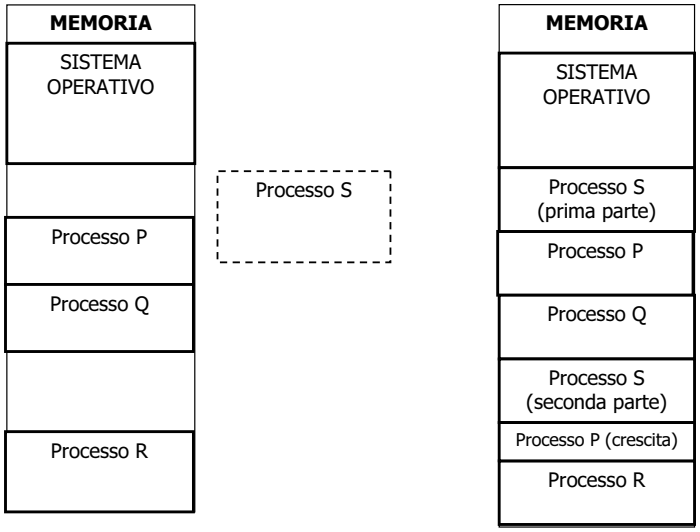
- Memoria fisica disponibile limitata
- **Buffers**: area di memoria allocata a contenere i dati in transito da disco (DMA) a memoria centrale
- LINUX tende a mantenere in memoria, il più a lungo possibile, i dati letti da disco e i dati scritti dai programmi per limitare gli accessi (lenti) a disco
- Necessità di liberare memoria fisica:
  - LINUX diminuisce la zona di memoria riservata ai buffers

LINUX e la memoria virtuale (2)

- Se non è sufficiente, è necessario rendere disponibili zone di memoria fisica che sono allocate a processi in esecuzione
  - se la zona di memoria da liberare contiene informazioni che non sono state modificate rispetto alla loro copia su disco (es. Codice) la zona di memoria viene considerata libera
  - se la zona di memoria da liberare contiene informazioni che sono state modificate, queste vengono salvate su disco (nello **Swap file**) per poter essere recuperate
- Condivisione di codice tra i processi (**sharing**): LINUX mantiene in memoria una sola copia del codice o delle librerie condivise da più processi
- Comando **free** per vedere la situazione della memoria

Rilocazione

- Tutti i meccanismi di gestione della memoria virtuale rilocano il programma come un insieme di blocchi di dimensione fissa (**pagine**), eliminando così la necessità di individuare un blocco di memoria contigua sufficientemente ampio da potervi caricare il programma
- Il sistema operativo ha solo bisogno di trovare un numero sufficiente di pagine (non necessariamente contigue) disponibili nella memoria fisica
- La paginazione ha inoltre l'effetto di ridurre il fenomeno della **frammentazione della memoria**: cioè della presenza di zone di memoria libere (e piccole) inframmezzate a zone utilizzate
- E' inoltre possibile gestire facilmente la crescita di memoria di un processo durante l'esecuzione



Paginazione (1)

- Lo spazio di **indirizzamento virtuale** di ogni programma è **lineare** (indirizzi virtuali contigui) ed è suddiviso in un **numero** intero di **pagine** di dimensione fissa e potenza di 2
  - Dimensioni di pagina tipici: da 512 byte a 64 Kbyte
- L'indirizzo virtuale può essere visto come:

Numero Pagina Virtuale	Spiazzamento nella pagina

Paginazione (2)

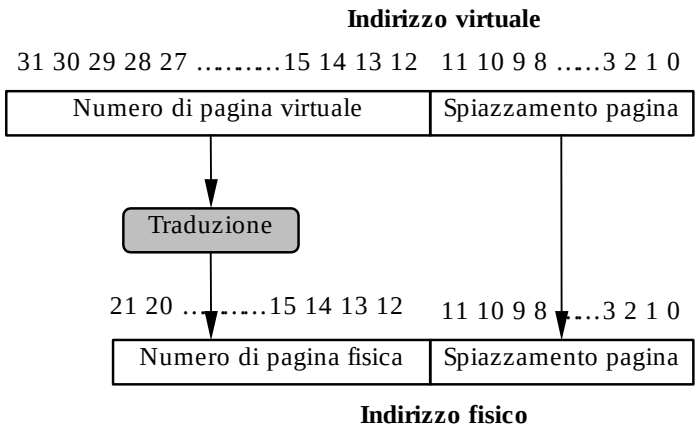
- Lo spazio di **indirizzamento fisico** (ossia della memoria centrale) viene suddiviso in un **numero** intero di **pagine** di uguale dimensione di quelle utilizzate per lo spazio di indirizzamento virtuale.
- Ogni pagina della memoria può quindi contenere esattamente una pagina dello spazio di indirizzamento virtuale
- L'indirizzo fisico può essere visto come

Numero Pagina Fisica	Spiazzamento nella pagina

Esempio

- Spazio di indirizzamento virtuale: indirizzi da 32 bit ⇨ 2^{32} indirizzi diversi
- Dimensione di pagina: 4K parole ⇨ 2^{12} indirizzi per lo spiazzamento
- Numero di pagine dello spazio di indirizzamento virtuale = $2^{32} / 2^{12} = 2^{20}$ pagine
- Spazio di indirizzamento fisico: 4M parole ⇨ 2^{22} indirizzi
- Numero di pagine dello spazio di indirizzamento fisico = $2^{22} / 2^{12} = 2^{10}$ pagine

Traduzione dell'indirizzo virtuale in indirizzo fisico



Paginazione (3)

- La corrispondenza tra pagine virtuali e pagine fisiche è generata da una **Tabella delle pagine** associata al processo
- La tabella delle pagine deve, in linea di principio, contenere una riga per ogni pagina virtuale del processo
- In questa ipotesi, il numero di pagina virtuale può essere utilizzato come indice nella tabella delle pagine del processo

Memoria virtuale di P

Numero di pagina	Contenuto delle pagine
0x00000	AAAA
0x00001	BBBB
0x00002	CCCC
0x00003	DDDD

Memoria virtuale di Q

Numero di pagina	Contenuto delle pagine
0x00000	RRRR
0x00001	SSSS
0x00002	TTTT
0x00003	UUUU
0x00004	VVVV

Tabella Pagine di P

NPV	NPF
0x0000**0**	0x0000**4**
0x0000**1**	0x0000**5**
0x0000**2**	0x0000**6**
0x0000**3**	0x0000**7**

Tabella Pagine di Q

NPV	NPF
0x0000**0**	0x0000**8**
0x0000**1**	0x0000**9**
0x0000**2**	0x0000**A**
0x0000**3**	0x0000**B**
0x0000**4**	0x0000**C**

MEMORIA FISICA

Numero di pagina	Contenuto delle pagine
0x0000**0**	S.O.
0x00001	S.O.
0x00002	S.O.
0x00003	S.O.
0x00004	AAAA
0x00005	BBBB
0x00006	CCCC
0x00007	DDDD
0x00008	RRRR
0x00009	SSSS
0x0000A	TTTT
0x0000B	UUUU
0x0000C	VVVV
0x0000D	non usata
0x0000E	non usata
0x0000F	non usata

Memory Management Unit

- La traduzione del numero di pagina virtuale nel corrispondente numero di pagina fisica avviene mediante un **dispositivo specializzato**, la Memory Management Unit (MMU)
- Può essere posizionata nel chip della CPU, o in un chip separato

MMU (1)

- La MMU ha al suo interno una memoria molto veloce (e di dimensioni ridotte) che contiene, in un formato opportuno, le tabelle delle pagine di tutti i processi
- In generale, la tabella delle pagine implementata nella MMU
  - contiene solo una **parte** della tabella delle pagine di ogni processo (e cioè le righe relative alle pagine maggiormente utilizzate). Il valore di **NPV** non può quindi essere utilizzato come indice della tabella ma è memorizzato come campo
  - per distinguere le pagine di ciascun processo, un campo della tabella contiene il **PID** del processo stesso

MMU (2)

PID corrente

NPV	Offset nella pagina

PID	NPV	NPF

- La memoria della MMU viene realizzata spesso come memoria **associativa**:
  - la selezione di una riga della memoria non avviene tramite indirizzo ma come associazione sul **contenuto** di opportuni campi (descrittore) della riga stessa
  - il descrittore utilizzato è la coppia (**PID, NPV**) per ottenere il numero di pagina fisica corrispondente (NPF). A tale valore viene giustapposto lo spiazzamento nella pagina completando la traduzione dell'indirizzo

MMU (3)

- La dimensione della tabella associativa è pari a
  - N° processi in esecuzione X **R**
  - dove **R** è il numero di pagine residenti in memoria centrale per ogni processo
- Si noti che il contenuto della tabella associativa è sempre una parte del contenuto delle tabelle dei processi. Queste ultime hanno una immagine completa in memoria, rappresentata da una struttura dati del Sistema Operativo più complessa rispetto a quella presente nella MMU

Condivisione delle pagine

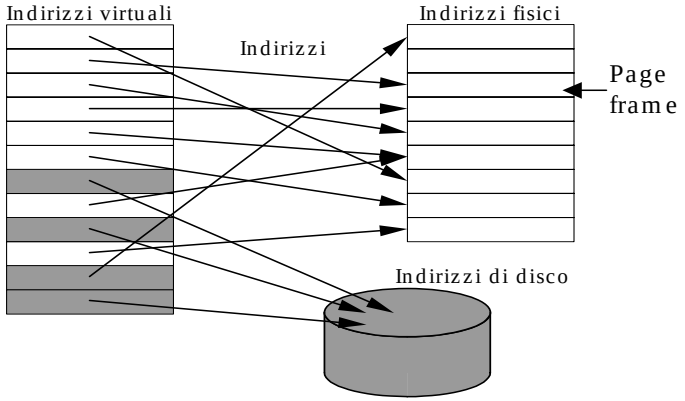
- I diversi processi possono condividere delle pagine (tipicamente di codice o di libreria, ma anche di dati, tramite opportuni meccanismi messi a disposizione dal S.O.). Le pagine condivise sono presenti una sola volta nella memoria fisica.
- Per ogni pagina condivisa, esiste una riga (cioè un NPV) nella corrispondente tabella delle pagine di ogni processo condividente.
- I valori di NPF relativi alle pagine condivise sono identici nelle righe di ogni processo che le condivide.

Protezione delle pagine

- Il meccanismo di paginazione consente di rilevare, durante l'esecuzione, un accesso a zone di memoria che non appartengono allo spazio di indirizzamento virtuale del processo in esecuzione.
  - In questo caso, la MMU (o il Sistema Operativo stesso) generano un interrupt di violazione di memoria
- E' possibile associare ad ogni pagina virtuale di un processo alcuni **bit di protezione**, che definiscono le modalità di accesso (diritti) consentite per quella pagina
  - Lettura (**R**)
  - Scrittura (**W**)
  - Esecuzione (**X**) per pagine di codice
- I diritti di accesso risultano particolarmente significativi nel caso di pagine condivise.
  - Questi bit devono essere gestiti dalla MMU che in caso di violazione genera un interrupt di violazione di memoria

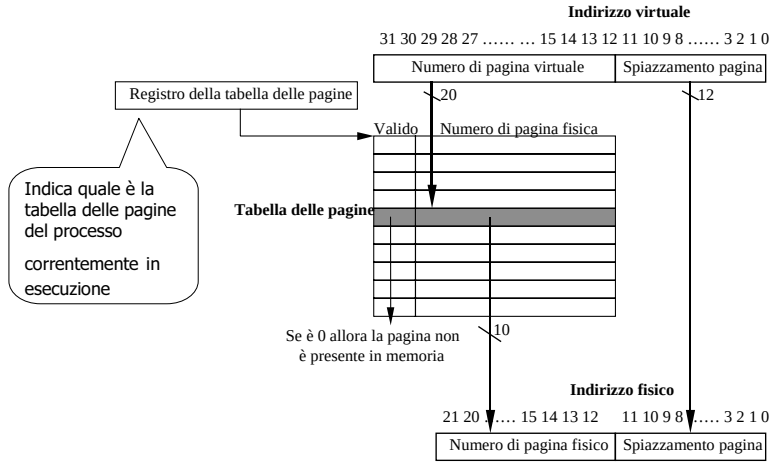
Pagine fuori memoria

- Non tutte le pagine virtuali di un processo sono caricate in memoria centrale. La tabella delle pagine può prevedere la presenza di un bit di **validità**



Costruzione dell'indirizzo fisico

Il bit **VALIDO** indica se la pagina corrispondente è presente/assente in memoria



Calcolo dell'indirizzo fisico (1)

- La MMU verifica se la pagina richiesta è presente in memoria centrale esaminando il bit di VALIDO (valid bit).
- Se il bit=1 la pagina è presente in memoria
  - Legge il valore contenuto nella tabella che rappresenta il numero di pagina fisica e lo affianca come bit più significativi all'offset ottenendo l'indirizzo fisico di memoria da cercare.
- Se il bit è a 0 si verifica un errore di pagina (**page fault**) e la pagina deve essere caricata da disco.

Calcolo dell'indirizzo fisico (2)

- Se la MMU contiene al suo interno solo le pagine residenti R di ogni processo, la mancanza della riga cercata nella tabella della MMU (fallimento della ricerca associativa) coincide con la mancanza della pagina nella memoria centrale (**table miss**)
- In caso di *table miss* si verifica quindi un errore di pagina (**page fault**)

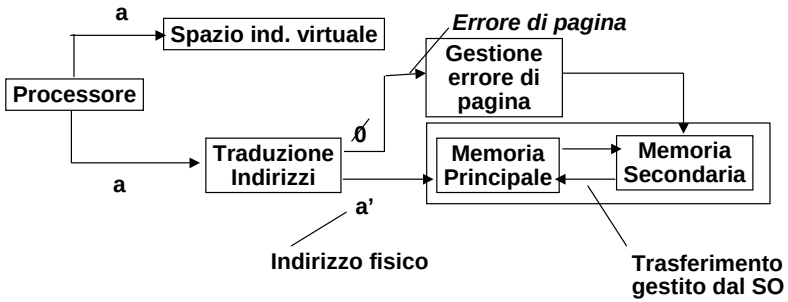
Traduzione degli indirizzi

V = {0, 1, . . . , n - 1} spazio indirizzi virtuali n > m
M = {0, 1, . . . , m - 1} spazio indirizzi fisici

MAP: V --> M U {Ø} f. traduzione degli indirizzi

MAP(a) = a' se il dato all'ind. virtuale a è presente all'ind. fisico a' e a' in M

= Ø se il dato all'ind. virtuale a non è presente in M



Gestione delle pagine virtuali non residenti in memoria

Durante l'esecuzione di un processo

- Un certo numero di pagine virtuali è caricato in memoria di lavoro in altrettante pagine fisiche (pagine residenti)
- Durante un accesso a memoria, in caso di page fault, tramite un interrupt di mancanza di pagina il **controllo** deve essere passato al **sistema operativo con il meccanismo dell'eccezione**.
- Il processo in esecuzione viene interrotto e il sistema operativo deve
  - rintracciare su disco la pagina virtuale richiesta: il S.O. utilizza per questo le tabelle delle pagine "complete" che contengono, per le pagine fuori memoria, il riferimento alla posizione su disco
  - trovare spazio in memoria per caricare la pagina richiesta. Questa operazione può implicare di dover scaricare da memoria un'altra pagina. Le tabelle delle pagine (e MMU) dei processi coinvolti devono essere aggiornate
  - caricare la pagina da disco
  - far rieseguire l'istruzione che aveva generato il page fault
- La condizione fondamentale per un **funzionamento efficiente** della memoria virtuale è che il numero di accessi che genera page fault sia basso (5%) rispetto al numero totale di accessi

Tabella delle pagine

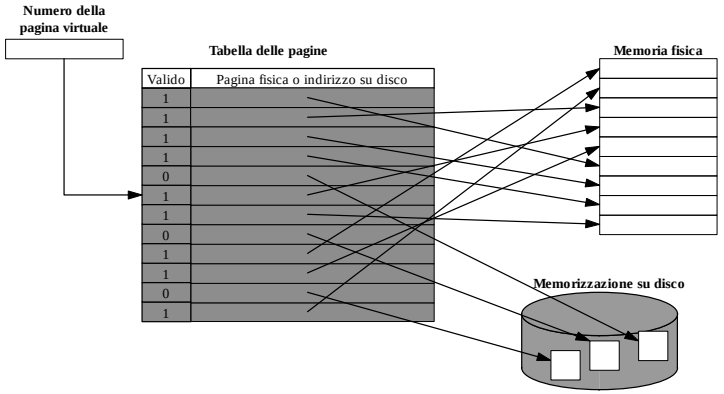


Tabella delle pagine e bit di controllo

- Per gestire in modo efficiente la scelta della pagina fisica in cui caricare la pagina che ha causato il page fault, ogni riga della tabella delle pagine può prevedere altri 2 bit di controllo:
  - **bit di accesso** (reference bit): che viene posto a 1 se si accede alla pagina fisica corrispondente (serve per gestire la sostituzione della pagine con algoritmo LRU);
  - **bit di modifica** (modify bit): che viene messo a 1 quando si accede in scrittura alla pagina fisica corrispondente (serve per gestire la ricopiatura della pagina in memoria su disco, quando questa viene scaricata).

Progetto di un sistema di memoria virtuale

- Problemi principali:
  1. **Dimensione delle pagine** (unità trasferita tra memoria principale e memoria secondaria)
  2. **Strategia di caricamento delle pagine** (su richiesta/working set)
  3. **Politica di sostituzione**: nel caso di errore di pagina, se la memoria centrale non ha page frame disponibile come scegliere la pagina da sostituire per far posto alla nuova pagina
  4. **Politica di posizionamento** delle pagine in memoria centrale

1. Dimensionamento delle pagine

- La dimensione di una pagina:
  - è costante;
  - è multipla di 2^n;
  - pagine di grandi dimensioni implicano un ridotto numero di pagine presenti ma anche uno sfruttamento della memoria non ottimale (slack area);
  - pagine di piccole dimensioni implicano un elevato numero di pagine e quindi di righe nella MMU. Inoltre:

Dimensionamento delle pagine

- Pagine piccole riducono l'efficienza dei trasferimenti dal disco:
  - Pagine più piccole richiedono più trasferimenti
  - Tempo di ricerca della pagina su disco + ritardo di rotazione ≈ 10 msec
  - Tempo di trasferimento di una pagina di 1KByte a 10MB/sec richiede circa 0.1 msec
  - Il peso del trasferimento è di due ordini di grandezza inferiore al tempo di ricerca

Dimensionamento delle pagine

- Pagine piccole portano a meno thrashing se il programma lavora su regioni piccole e separate all'interno dello spazio di indirizzamento
- Esempio: matrice di 10.000×10.000 elementi memorizzata per colonne, ogni elemento richiede 8 Byte.
- Gli elementi della riga 1 sono a distanza di 80.000 Byte l'uno dall'altro.
- Elaborazione su una riga intera: per avere tutti gli elementi in memoria:
  - Con pagine da 8KB sono necessari 80MB
  - Con pagine da 1KB sono necessari 10MB
  - Se memoria disponibile è di 32 MB il programma andrebbe in thrashing con pagine da 8KB.

2. Caricamento delle pagine

- Sono presenti due strategie:
  - paginazione su richiesta;
  - working set.

2.a Paginazione su richiesta

- Esecuzione di un nuovo programma
  - La tabella delle pagine del processo ha tutti i bit Valid a 0 (nessuna pagina si trova in memoria)
  - Quando la CPU cerca di accedere alla prima istruzione si verifica un errore di pagina e la prima pagina viene portata in memoria e registrata nella tabelle delle pagine
  - Ogni volta che si identifica un indirizzo in una pagina non ancora in memoria, si verifica un errore di pagina
- Metodo chiamato di **paginazione su richiesta (demand paging):** pagine caricate solo quando necessarie
  - Questo metodo viene utilizzato per il caricamento iniziale del programma

Caratteristiche dell'accesso a memoria dei programmi

- La distribuzione degli accessi a memoria nello spazio e nel tempo non è omogenea, ma presenta la caratteristica di **località**
- **Località temporale**: indica che un programma tende ad accedere, in un certo intervallo di tempo, agli indirizzi di memoria a cui ha acceduto di recente (cicli)
- **Località spaziale**: indica che un programma accede con maggior probabilità agli indirizzi vicini a quelli che ha referenziato di recente (istruzioni sequenziali, strutture dati lineari)

2.b Working set (1)

- Metodo basato sul principio di località
  - La maggior parte dei programmi non accede al suo spazio di indirizzamento in modo uniforme ma gli accessi tendono a raggrupparsi in un numero limitato di pagine.
- **Working set**: insieme di pagine che ad ogni istante t rappresentano le pagine a cui hanno fatto accesso le ultime k richieste alla memoria. Se k è sufficientemente grande, il working set di un programma varia molto lentamente.

Working set (2)

- Il numero R di pagine residenti è ottenuto da una stima del Working set (in configurazione)
- Due esigenze contrastanti:
  - R grande: pochi page fault
  - R limitato: molti processi in memoria
- All'inizio dell'esecuzione di un programma, tramite *demand paging*, è possibile caricare il working set in memoria (**R** page fault)

3 Politica di sostituzione delle pagine

- Definizione di un algoritmo di sostituzione
  - Casuale
  - Least recently used (LRU): il SO cerca di identificare la pagina meno utile nel prossimo futuro, ossia quella utilizzata meno di recente (principio di località) che ha bassa probabilità di appartenere al working set attuale
  - FIFO (First In First Out): si sostituisce sempre la pagina caricata meno di recente, indipendentemente da quando si è fatto riferimento a questa pagina

Algoritmo LRU

- Buone prestazioni in media
- Necessari dei bit di controllo nella tabella della pagine per mantenere l'informazione degli accessi
- Per semplificare: un solo **bit di accesso** che viene posto a 1 quando la pagina viene referenziata e viene azzerato periodicamente dal SO.
  - Durante l'azzeramento periodico, il SO incrementa una variabile di conteggio interna, per tutte le pagine con bit di accesso a 0.
  - Viene sostituita la pagina con bit di accesso a 0 e valore di conteggio più alto
- Esistono situazioni particolari in cui l'algoritmo ha prestazioni pessime

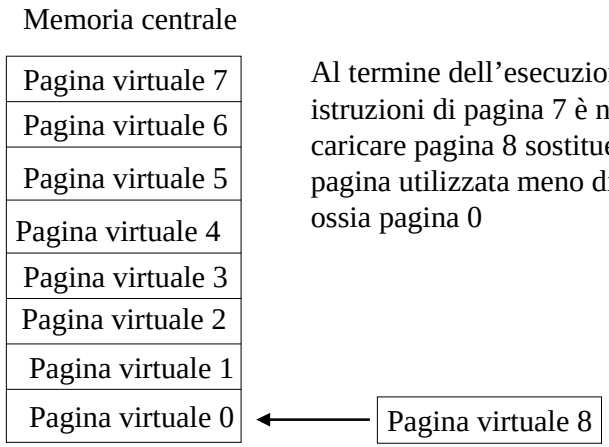
Esempio di applicazione

- Esempio:
  - Esecuzione di un ciclo che si estende su 9 pagine virtuali
  - La memoria può contenere solo 8 pagine
  - Si assuma che le prime 8 pagine virtuali (da 0 a 7) siano in memoria.
  - Al termine dell'esecuzione della pagina 7 è necessario caricare pagina 8.

Algoritmo LRU: esempio

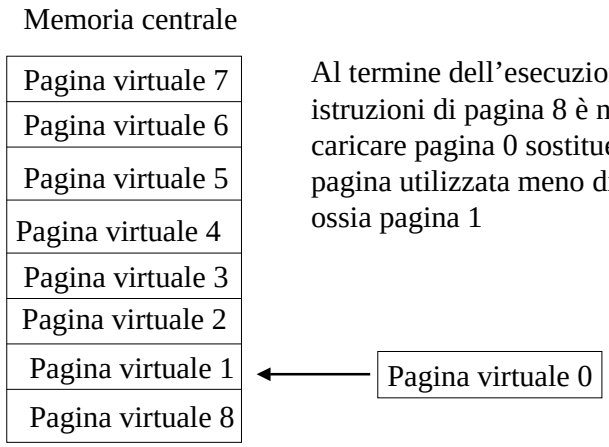
- Scelta della pagina da sostituire mediante algoritmo LRU
- La pagina meno utilizzata di recente è la pagina 0 ➔ viene sostituita con la pagina 8
- Al termine dell'esecuzione delle istruzioni che appartengono alla pagina 8 è necessario ricominciare l'esecuzione del ciclo da pagina 0, ossia la pagina appena sostituita ➔ errore di pagina!
- La pagina 0 andrà a sostituire la pagina 1, utilizzata meno di recente: ma questa sarà la pagina da ricaricare subito dopo!!

Fallimento dell'algoritmo LRU



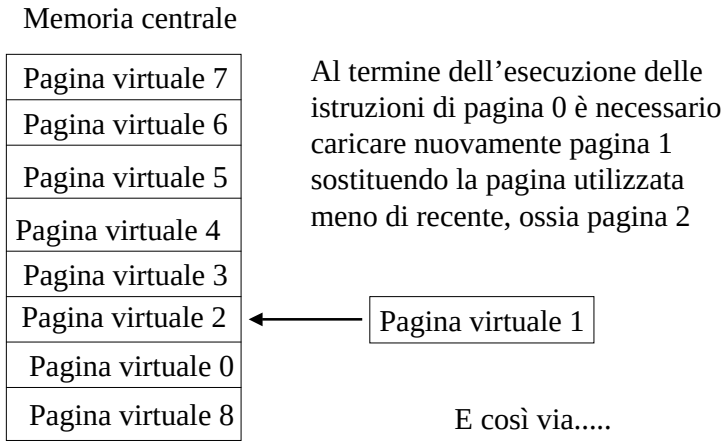
Al termine dell'esecuzione delle istruzioni di pagina 7 è necessario caricare pagina 8 sostituendo la pagina utilizzata meno di recente, ossia pagina 0

Fallimento dell'algoritmo LRU



Al termine dell'esecuzione delle istruzioni di pagina 8 è necessario caricare pagina 0 sostituendo la pagina utilizzata meno di recente, ossia pagina 1

Fallimento dell'algoritmo LRU



Algoritmo LRU

- Fallisce quando il working set è di poco maggiore della memoria centrale disponibile
- Nel caso in esempio il working set è di una pagina superiore al numero di pagine disponibili in memoria
- Se l'accesso alle pagine è sequenziale, in questo caso si sostituisce sempre la pagina che verrà richiesta subito dopo

Algoritmo di sostituzione FIFO

- Si mantiene l'informazione del momento in cui è stata caricata la pagina
- Si associa un contatore ad ogni pagina fisica
- Inizialmente i contatori sono = 0
- Al termine della gestione di un errore di pagina i contatori delle pagine fisiche non modificati sono incrementati di 1
- Il contatore della pagina appena caricata viene posto a 0

Algoritmo di sostituzione FIFO

- In caso di errore di pagina la pagina da sostituire è quella con il valore del contatore più alto.
- Il valore di contatore più alto significa che si tratta della pagina che ha visto piiù errori di pagina e quindi si trova in memoria da più tempo rispetto alle altre
  - ⇨ ha più probabilità di non essere necessaria nel prossimo futuro (principio di località)

Algoritmi di sostituzione

- L'algoritmo FIFO non ha un buon comportamento nel caso in cui il working set è di poco maggiore del numero di pagine disponibili in memoria
- Non ci sono algoritmi che danno buoni risultati in questo caso
- Un programma che genera errori di pagina frequenti si dice in thrashing (situazione poco auspicabile)

4. Scrittura in memoria

- Nel caso di scrittura in memoria la modifica avviene solo nelle pagine in memoria e non riportata immediatamente sulle pagine su disco.
- Motivazione: tempi di accesso al disco molto elevati rispetto ai tempi di accesso alla memoria.
- In caso di errore di pagina è necessario individuare se la pagina da scaricare è stata o meno modificata.

Scrittura in memoria

- Se la pagina è stata modificata è necessario copiarla su disco prima di sostituirla.
- Le pagine contenenti istruzioni non possono essere modificate quindi non devono essere copiate su disco.
- Le pagine contenenti dati e/o allocate dinamicamente devono essere salvate nello **swap file**.
- Nella MMU si associa un bit (**bit di modifica**) ad ogni pagina fisica per indicare se la pagina che contiene è stata o meno modificata.
- Quando una pagina deve essere sostituita il SO verifica il bit di scrittura per stabilire se la pagina debba essere copiata su disco.

Frammentazione della memoria

- Ad un programma e ai suoi dati vengono assegnati un numero intero di pagine
- Non sempre un programma e i suoi dati riempiono un numero intero di pagine
  - ⇨ spreco della memoria
- Esempio: programma + dati = 26.000 Byte e pagine di 4kByte (4096 Byte)
- Numero di pagine: 26000/4096 ≈ 6.34
  - ⇨ sono necessarie 7 pagine, ma l'ultima pagina è occupata per 1424 Byte e verranno sprecati 2672 Byte

Frammentazione della memoria

- Per pagine di n Byte la quantità media di spazio di memoria sprecato nell'ultima pagina sarà di n/2 Byte
  - ⇨ ridurre le dimensioni della pagina per minimizzare lo spreco
  - Pagine piccole ⇨ ↑ numero di pagine
  - ⇨ ↑ dimensioni della tabella delle pagine
  - ⇨ ↑ dimensioni della MMU

Strutturazione della memoria virtuale **Segmentazione** (1)

- Nella paginazione, per ogni processo, si considera un unico spazio di indirizzamento virtuale lineare e indifferenziato
- Lo spazio di indirizzamento virtuale può essere diviso in un insieme di **spazi virtuali indipendenti** dal punto di vista logico. Ad es. codice, dati, stack..
- Gli spazi di indirizzamento virtuali indipendenti vengono chiamati **segmenti**.

Strutturazione della memoria virtuale **Segmentazione** (2)

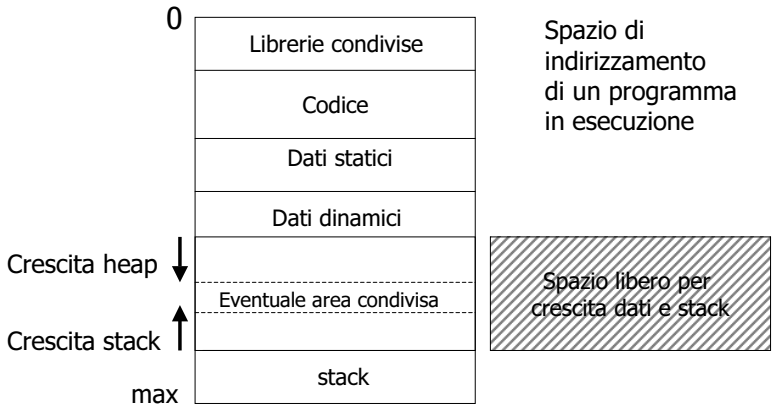
- Ogni segmento composto da una sequenza lineare di indirizzi.
- Ogni segmento è caratterizzato da una coppia di indirizzi virtuali che ne definiscono l'inizio e la fine
- Poichè la segmentazione delle aree virtuali si appoggia alla paginazione della memoria fisica è sufficiente avere un $NPV_{iniziale}$ e un NPV_{finale}
- Segmenti

Codice	Dati dinamici (heap)	Condivisa (Codice, ma anche dati tramite opportune dichiarazioni fornite dal S.O.)
Dati Statici	Stack	Librerie dinamiche

Segmenti - Vantaggi

- E' possibile definire diversi diritti di accesso a seconda del segmento
- I segmenti possono modificare la loro lunghezza in fase di esecuzione (ad esempio lo stack o i dati dinamici) e ogni segmento può crescere indipendentemente, senza creare problemi ad altri segmenti
- E' possibile individuare aree di memoria "logiche" da condividere
- Semplifica la gestione del collegamento di procedure compilate separatamente, ognuna in un segmento separato

Struttura dello spazio di indirizzamento virtuale di un programma segmentato (exec)



Crescita delle aree dati dinamiche

- Lo stack cresce automaticamente ad ogni attivazione di funzione, senza istruzioni o funzioni di sistema esplicite
- Lo heap cresce tramite invocazione esplicita di servizi di sistema (la funzione *malloc* può chiamare al suo interno tali servizi)
- Il servizio di sistema LINUX che incrementa lo heap è la funzione

  `* void sbrk (int incremento)`
- che incrementa l'area di heap di un n° di pagine pari all'intero superiore di (incremento/dimensione della pagina) e restituisce l'indirizzo iniziale della nuova area

Confronto tra paginazione e segmentazione

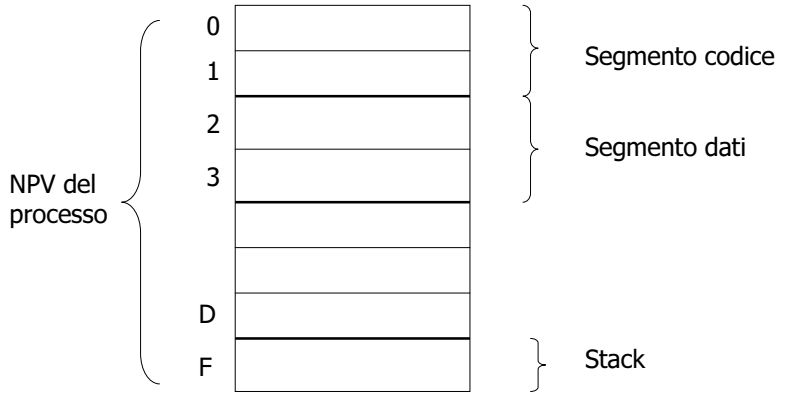
	Paginazione	Segmentazione
Visibilità al programmatore	NO	SI
Spazi di indirizzamento	1	Molti
Spazio virtuale > spazio fisico di memoria?	SI	SI
Motivo della tecnica?	Simulare memorie molto grandi	Fornire spazi multipli di indirizzamento

Segmentazione - struttura dell'indirizzo virtuale all'interno di un segmento

- Lo spazio di **indirizzamento virtuale** di **ogni segmento** è **lineare** (indirizzi virtuali contigui) ed è suddiviso in un **numero** intero di **pagine** di dimensione fissa
- Quindi all'**interno di ogni segmento** l'indirizzo virtuale può essere visto come

Numero pagina virtuale nel segmento	Spiazzamento nella pagina

Segmentazione - rappresentazione dello spazio di indirizzamento virtuale del processo



- Lo spazio di indirizzamento virtuale di un processo è rappresentato in segmenti contigui, suddivisi in pagine, il cui riferimento è assoluto rispetto al processo

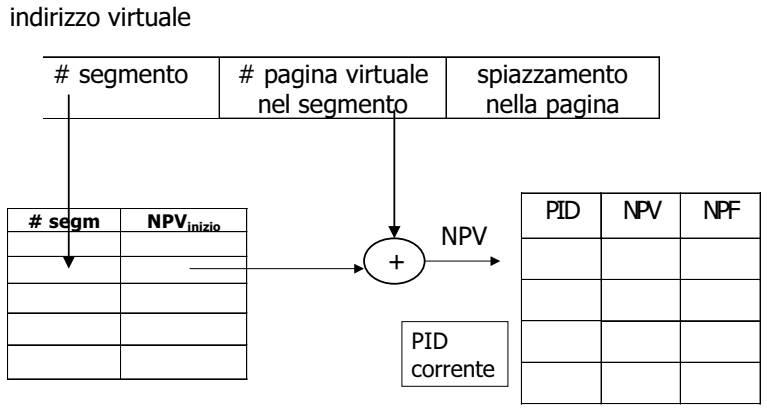
Segmentazione: struttura dell'indirizzo fisico

- Lo spazio di **indirizzamento fisico** (ossia della memoria centrale) viene suddiviso in un **numero** intero di **pagine** di uguale dimensione di quelle utilizzate per lo spazio di indirizzamento virtuale
- Ogni pagina della memoria può quindi contenere esattamente una pagina dello spazio di indirizzamento virtuale
- La **MMU** traduce in modo automatico l'indirizzo virtuale in indirizzo fisico tramite opportune tabelle

Tabelle per la segmentazione

- Per la traduzione dell'indirizzo virtuale in indirizzo fisico è necessaria:
- una **tabella dei segmenti per ogni processo** (una riga per ogni segmento, ogni riga contiene la pagina virtuale d'inizio del segmento stesso)
- **la tabella delle pagine complessiva per ogni processo** (con una riga per ogni pagina dello spazio di indirizzamento virtuale). Questa tabella ha la stessa struttura di quella della paginazione e l'insieme delle tabelle dei processi possono essere implementate in modo più efficiente in MMU con una tabella associativa

Segmentazione: calcolo dell'indirizzo



Segmentazione: calcolo dell'indirizzo

# segmento	# pagina virtuale nel segmento	Spiazzamento nella pagina

- Dato un processo in esecuzione (**PID**), un indirizzo virtuale e il segmento cui appartiene:
- con **#segmento** si accede alla tabella dei segmenti del processo e si ottiene il **# pagina virtuale d'inizio** del segmento
- **# pagina virtuale d'inizio** + **# pagina virtuale nel segmento = NPV**
- con **PID**, **NPV** si accede alla tabella associativa per recuperare **NPF**

Esempio

Condizioni iniziali

- Due processi P e Q
- Spazio di indirizzamento virtuale per ogni processo: 64K
- Dimensione delle pagine 4K
- Spazio di indirizzamento fisico 16M

Struttura degli indirizzi

- indirizzo virtuale:
  - 16 bit: 12 per offset nella pagina e 4 per NPV (16 pagine virtuali per ogni processo)
- indirizzo fisico
  - 24 bit, 12 per offset nella pagina e 12 per NPF (4K pagine fisiche in memoria centrale)

Caratteristiche iniziali dei segmenti

Processo P

- CP (codice) = **7K** (condiviso)
- DP (dati) = **9K**
- PP (pila) = **4K**
- COND (dati condivisi) = **6K**

Processo Q

- CQ (codice) = **7K** (condiviso)
- DQ (dati) = **19K**
- PQ (pila) = **4K**
- COND (dati condivisi) = **6K**

- In entrambi i processi il segmento COND è allocato lasciando 3 pagine libere dopo il segmento dati iniziale per permettere la crescita dello heap

Spazio di indirizzamento virtuale e pagine virtuali

	Processo P	Processo Q
0	**CP0**	**CQ 0**
1	**CP1**	**CQ 1**
2	**DP0**	**DQ0**
3	**DP1**	**DQ1**
4	**DP2**	**DQ2**
5		**DQ3**
6		**DQ4**
7		
8	**COND0**	
9	**COND1**	
A		**COND0**
B		**COND1**
C		
D		
E		
F	**PP0**	**PQ0**

LINUX e la gestione della memoria (1)

- Fork ( )
  - L'esecuzione della fork crea un nuovo processo che è l'immagine del padre
  - Viene aggiornata la tabella dei processi, ma non viene allocata memoria fisica e i segmenti virtuali vengono condivisi
  - All'atto di una scrittura in memoria, da parte di uno dei due processi, la pagina scritta viene effettivamente allocata (le pagine relative a segmenti virtuali contenenti dati vengono inizialmente poste con diritto di accesso in sola lettura)

LINUX e la gestione della memoria (2)

- Exec ( )
  - L'esecuzione della exec pone a NON VALIDO tutte le pagine relative a codice e dati del processo
  - Tramite la tecnica di *demand paging* vengono caricate le pagine di codice e dati del programma lanciato in esecuzione

LINUX e la gestione della memoria (3)

Nel caso di page fault o errore di accesso ad una pagina virtuale V

- Se la pagina V **appartiene** allo spazio di indirizzamento virtuale del processo, LINUX verifica i diritti di accesso:
  - se l'accesso è legittimo (*page fault*), viene invocata la routine *swap_in* di caricamento della pagina da disco in memoria centrale
  - se l'accesso non è legittimo
    - se violazione in scrittura dopo fork, allora viene creata e allocata una copia della pagina
    - altrimenti il programma viene abortito

LINUX e la gestione della memoria (4)

- Se la pagina V **non appartiene** allo spazio di indirizzamento virtuale del processo:
  - se si tratta di crescita dello stack, una nuova pagina viene allocata per lo stack
  - altrimenti il programma viene abortito
- Se per l'allocazione di una nuova pagina in memoria non esistono pagine libere, LINUX invoca la routine *swap_out* di scaricamento di una pagina secondo un algoritmo basato sulla tecnica LRU