Lezione 16 [04/11/22]

Capitolo 8: Gestione della memoria

- Introduzione
- Avvicendamento dei processi (swapping)
- Allocazione contigua della memoria
- Segmentazione
- Paginazione
- Struttura della tabella delle pagine
- Esempio: le architetture Intel a 32 e 64 bit
- Esempio: architettura ARM

@redyz13 e @rosacarota

Introduzione

La memoria consiste in un grande **vettore di byte**, ciascuno con il proprio indirizzo

La CPU preleva le istruzioni dalla memoria sulla base del contenuto del **contatore di programma**, tali istruzioni possono determinare ulteriori letture **(load)** e scritture **(store)** in specifici indirizzi di memoria

Gli unici dispositivi di memoria accessibili dalla CPU sono la memoria centrale ed i registri interni alla stessa CPU

Esistono istruzioni macchina che accettano indirizzi di memoria come argomenti: nessuna istruzione accetta indirizzi di un disco

Tutte le istruzioni in esecuzione, unitamente ai dati devono essere trasferite in memoria prima che la CPU possa operare su di essi

Un programma per essere eseguito deve essere caricato nella memoria ed inserito all'interno di un processo

Durante la propria esecuzione un processo deve risiedere, almeno parzialmente, in memoria centrale

Ci saranno contemporaneamente più processi in memoria

Coda d'ingresso (input queue):

- L'insieme dei processi presenti nei dischi e che attendono di essere trasferiti nella memoria per essere eseguiti

Protezione della memoria

fredyz13 e lirosacarota

Per poter avere più processi in memoria è necessario assicurarsi che ciascun processo abbia uno spazio di memoria separato

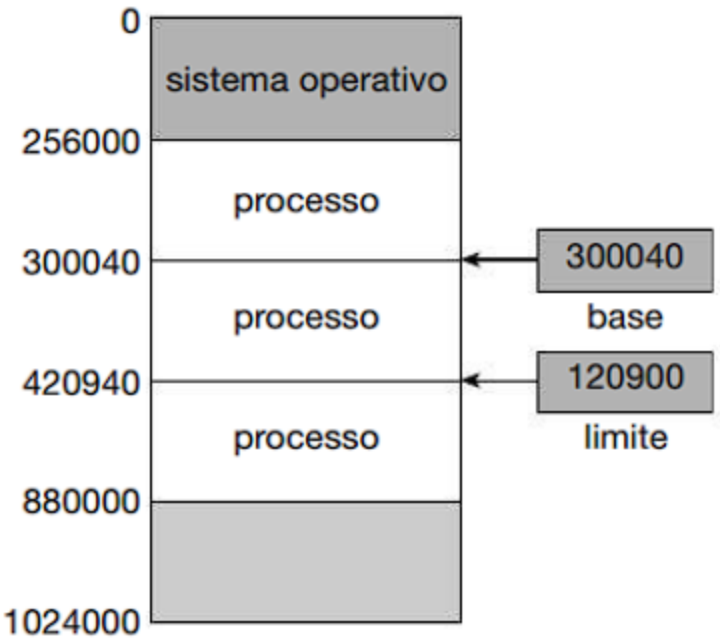
Per separare lo spazio di memoria dei programmi serve la capacità di determinare l'intervallo di indirizzi cui il programma può accedere

Due registri determinano il range di indirizzi legali a cui un programma può accedere:

- **Registro di base (base register):** contiene il più basso indirizzo della memoria fisica al quale il programma può accedere
- **Registro di limite (limit register):** contiene la dimensione dell'intervallo ammesso

La memoria al di fuori dell'intervallo individuato dai due registri non deve essere accessibile al programma

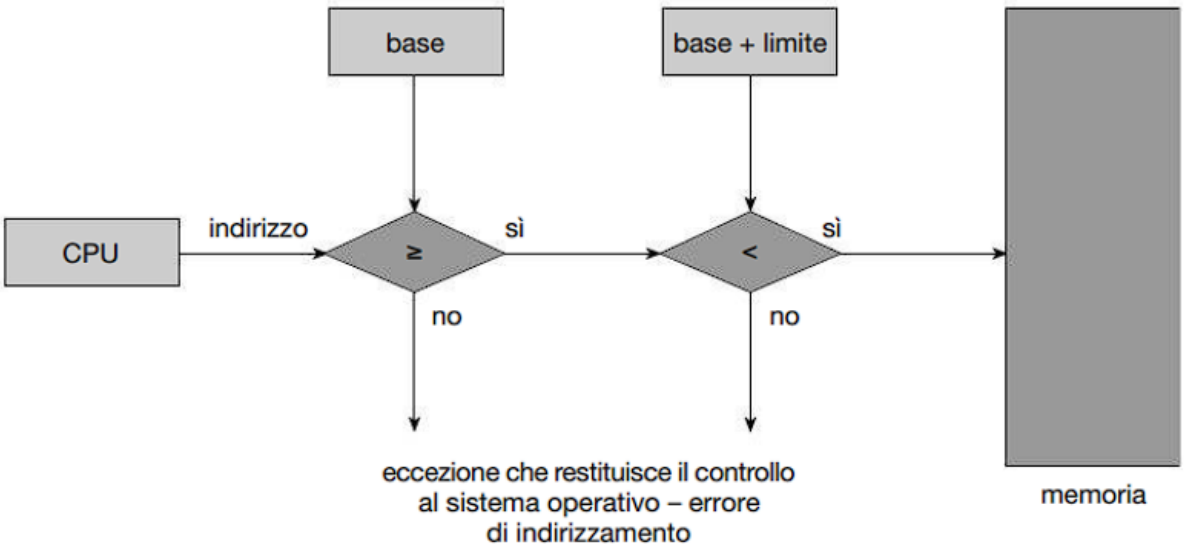
Uso di un registro di base e di un registro limite:



Funzionando in modalità kernel, il S.O. può accedere sia alla memoria ad esso riservata sia a quella riservata agli utenti

Le istruzioni di caricamento dei registri di base e di limite devono essere **istruzioni privilegiate** (possono essere eseguite solo in modalità kernel)

Architettura di protezione degli indirizzi con registri di base e di limite:

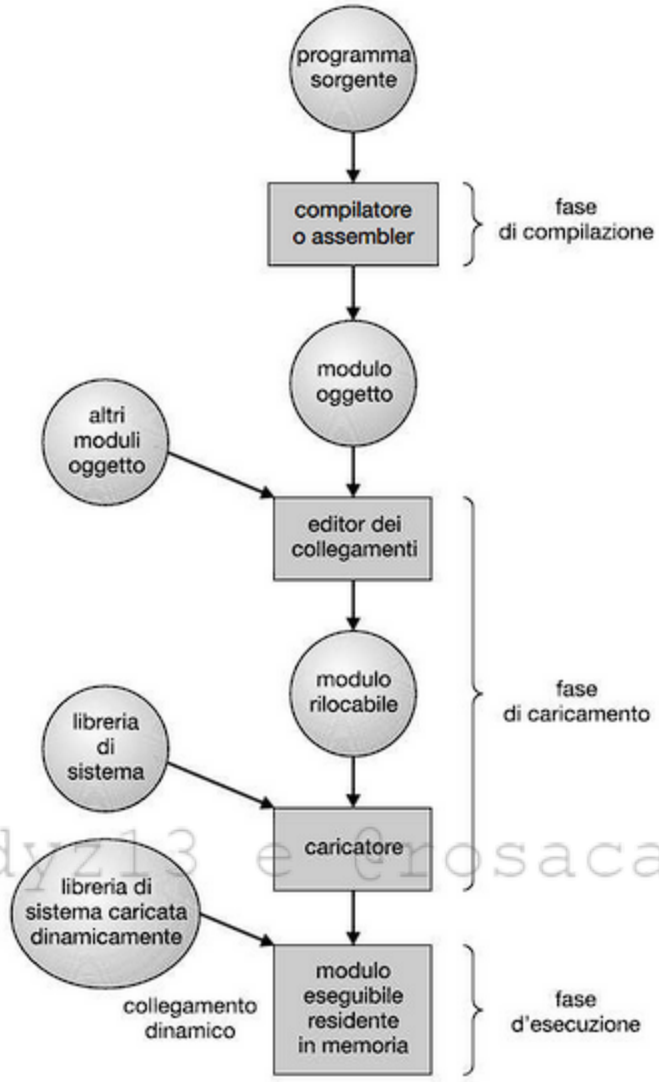


La CPU confronta ciascun indirizzo generato dai processi in modalità utente con i valori contenuti nei due registri

Qualsiasi tentativo da parte di un programma eseguito in modalità utente di accedere alle aree di memoria riservate al S.O. o riservate ad altri utenti comporta l'invio di una **eccezione (trap)** che restituisce il controllo al sistema operativo

Fasi di elaborazione per un programma utente:

@redyz13 e @rosacarota



@redyz13 e @rosacarota

Un programma utente, prima di essere eseguito, passa attraverso vari stati in cui gli indirizzi possono essere rappresentati in modi diversi

Associazione di istruzioni e dati ad indirizzi di memoria

La procedura normale consiste nel selezionare uno dei processi dalla coda di input e caricarlo in memoria

Durante la propria esecuzione il processo può accedere a istruzioni e dati in memoria

Quando il processo termina, la memoria che gli era stata allocata, viene deallocata

La maggior parte dei sistemi consente ai processi utenti di risiedere in qualsiasi parte della memoria fisica

Generalmente gli indirizzi del programma sorgente sono **simbolici** (per esempio il nome di una variabile)

Un compilatore di solito associa **(bind)** questi indirizzi simbolici a **indirizzi rilocabili**

Il **linkage editor** o il **loader** fa corrispondere questi indirizzi rilocabili ad **indirizzi assoluti**

L'associazione di istruzioni e dati ad indirizzi di memoria si può compiere in fase di:

@redyz13 e @rosacarota

- **Compilazione:** se al momento della compilazione è possibile sapere dove il processo risiederà in memoria, può essere generato un **codice assoluto** (se è noto a priori che un processo utente inizia alla locazione r, anche il codice generato dal compilatore comincia da quella locazione). Se, in un momento successivo, la locazione iniziale dovesse cambiare allora sarebbe necessario ricompilare il codice (metodo non utilizzato da sistemi general purpose)
- **Caricamento:** se al momento della compilazione non è possibile sapere in che punto della memoria risiederà il processo, il compilatore deve generare un **codice rilocabile**. Così il collegamento finale viene ritardato fino al momento del caricamento e in caso l'indirizzo iniziale cambiasse sarebbe sufficiente ricaricare il codice utente ed incorporare il valore modificato
- **Esecuzione:** il processo, in questa fase, può essere spostato da un segmento all'altro della memoria a condizione che il

collegamento venga ritardato fino al momento dell’esecuzione del programma. Per realizzare questo è necessario disporre di hardware specializzato (metodo più spesso utilizzato nei sistemi general purpose)

Spazi di indirizzi logici e fisici

Il concetto di spazio degli indirizzi logici associato a uno spazio degli indirizzi fisici separato è fondamentale per una corretta gestione della memoria:

- **Indirizzo logico:**
  - Generato dalla CPU
- **Indirizzo fisico:**
  - Indirizzo visto dall’unità di memoria, cioè caricato nel **registro dell’indirizzo di memoria (memory address register - MAR)**

@redy713 e @rosacarota

L’insieme di tutti gli indirizzi logici generati da un programma è lo **spazio degli indirizzi logici**

L’insieme degli indirizzi fisici corrispondenti a tali indirizzi logici è lo **spazio degli indirizzi fisici**

Lo spazio degli indirizzi logico e fisico:

- Coincidono nell’associazione di istruzioni e dati ad indirizzi di memoria in fase di compilazione o caricamento
- Differiscono nell’associazione di istruzioni e dati ad indirizzi di memoria in fase di esecuzione; in questo caso ci si riferisce agli indirizzi logici con il termine di **indirizzi virtuali**

Unità di gestione della memoria (Memory Management Unit - MMU)

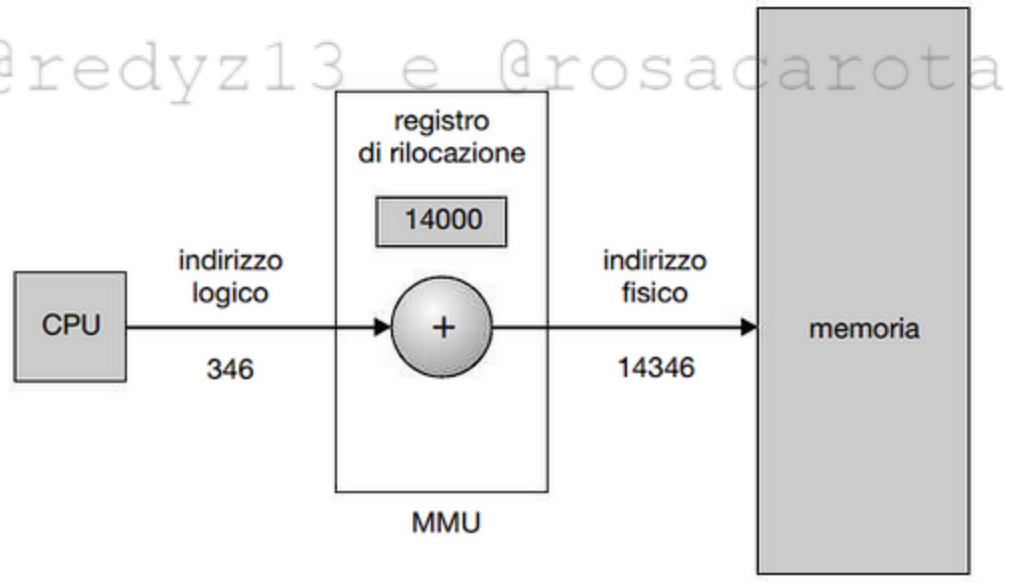
Unità di gestione della memoria (Memory Management Unit - MMU): dispositivo hardware che associa in fase di esecuzione gli indirizzi virtuali agli indirizzi fisici

In uno schema basato su MMU, il valore del registro di base, ora denominato **registro di rilocazione**, viene sommato ad ogni indirizzo generato da un processo utente, prima dell'invio all'unità di memoria

Il programma utente tratta esclusivamente indirizzi logici, non "vede" mai gli indirizzi fisici

Rilocazione dinamica tramite un registro di rilocazione:

@redyz13 e @rosacarota



- Intervallo degli indirizzi logici: $[0, max]$
- Intervallo degli indirizzi fisici: $[r + 0, r + max]$ per un valore base di r

Il programma utente pensa che il processo sia eseguito nell’intervallo degli indirizzi logici

Caricamento dinamico (dynamic loading)

Per migliorare l’utilizzo della memoria è possibile adoperare il caricamento dinamico: una routine viene caricata solo quando viene richiamata; tutte le procedure si tengono su disco in un formato di caricamento rilocabile

Si carica il programma principale in memoria e quando, durante l’esecuzione una procedura deve richiamarne un’altra, controlla innanzitutto che sia stata caricata:

- Se non è stata caricata, si richiama il linking loader rilocabile per caricare in memoria la procedura richiesta e aggiornare le tabelle degli indirizzi del programma in modo che registrino questo cambiamento. A questo punto il controllo passa alla procedura appena caricata

@wedyz13 e @rosacarota

Fornisce un migliore utilizzo dello spazio di memoria: le routine non utilizzate non verranno caricate

Utile quando sono presenti grandi quantità di codice che sono necessarie per la gestione di casi non frequenti

Non richiede un intervento particolare del sistema operativo, ma va progettato dall’utente in fase di progettazione di programma

Collegamento dinamico a librerie condivise (dynamic linking)

In un programma vengono spesso adoperate delle funzioni di libreria

Alcuni sistemi operativi supportano solo un **collegamento statico (static linking)**, nel quale le librerie sono trattate come

qualsiasi altra routine e vengono combinate dal loader nell'immagine binaria del programma

Con il **collegamento dinamico (dynamic linking)** viene posticipato il collegamento alle librerie fino al momento dell'esecuzione

Piccole porzioni di codice di riferimento, dette **stub**, indicano come localizzare la giusta procedura di libreria residente nella memoria o come caricare la libreria se non è presente la procedura

Lo stub poi rimpiazza se stesso con l'indirizzo della procedura per permettere l'esecuzione

Questo metodo è utile anche in caso di aggiornamento di librerie: nel caso di nuove versioni, se non ci fosse il linking dinamico tutti i programmi dovrebbero essere nuovamente linkati

Nel caso di nuove versioni di libreria, è importante che i programmi non utilizzino nuove versioni di libreria non compatibili; proprio per questo, sia nel programma che nella libreria sono aggiunte informazioni circa la versione

Fredyz13 e Rosacarota

I programmi linkati prima della variazione della libreria, continuano a servirsi della vecchia versione

Questo sistema è noto come **librerie condivise**

Il linking dinamico e le librerie condivise, a differenza del caricamento dinamico, necessitano di un supporto attivo del sistema operativo. Infatti, il S.O. è l'unico che può controllare se la routine richiesta da un processo è nella memoria di un altro processo o se più processi possono accedervi

Sovrapposizione di sezioni (overlay)

Per permettere a un processo di essere più grande della memoria che gli si assegna, si può utilizzare una tecnica chiamata **sovrapposizione di sezioni (overlay)**

Esso fornisce la possibilità di mantenere in memoria soltanto le istruzioni e i dati che si usano con maggior frequenza

Quando sono necessarie altre istruzioni queste si caricano nello spazio precedentemente occupato dalle istruzioni che non sono più in uso

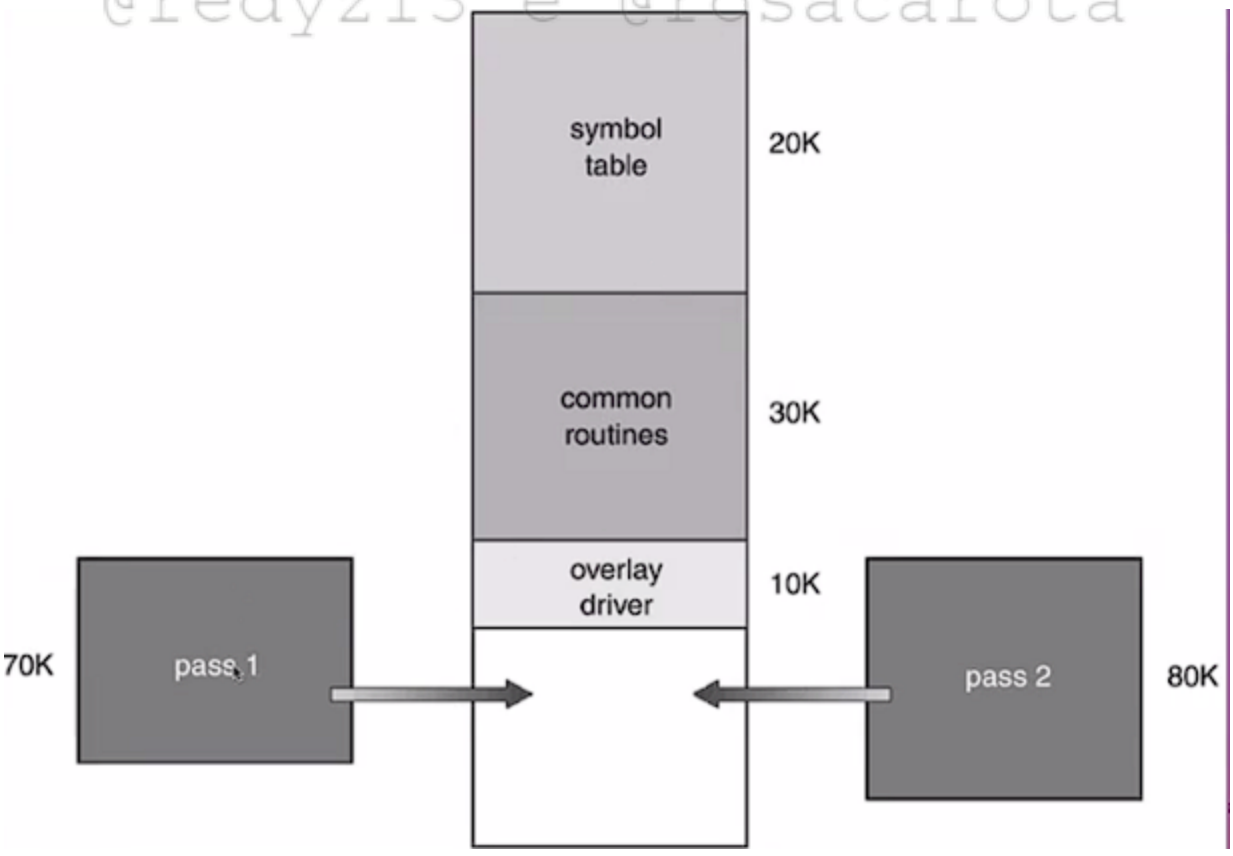
Questa tecnica è generalmente implementata dal programmatore per mezzo di strutture di file, copiandone il contenuto nella memoria e quindi trasferendo il controllo a quest'ultima per eseguire le istruzioni appena lette. Il SO si limita ad annotare la presenza di operazioni di I/O supplementari

Non richiede alcun intervento da parte del sistema operativo

Utilizzato nei microcalcolatori e in sistemi dotati di memoria limitata

Sovrapposizione di sezioni per un assemblatore a due passi:

@redyz13 e @rosacarota



Avvicendamento di processi (swapping)

Per essere eseguito un processo deve trovarsi nella memoria centrale

Un processo può essere trasferito temporaneamente nella **memoria ausiliaria (backing store)**, da cui si riporta nella memoria centrale al momento di riprendere l'esecuzione; ad es. in una situazione di time sharing ci sono normalmente molti utenti e la memoria centrale non è sempre sufficiente a contenere tutti i lori processi

Diventa necessario memorizzare i processi in eccesso nella memoria secondaria di supporto

Per eseguire questi processi bisogna spostarli dal disco alla memoria e viceversa

Questo metodo viene detto **avvicendamento dei processi nella memoria (swapping)**

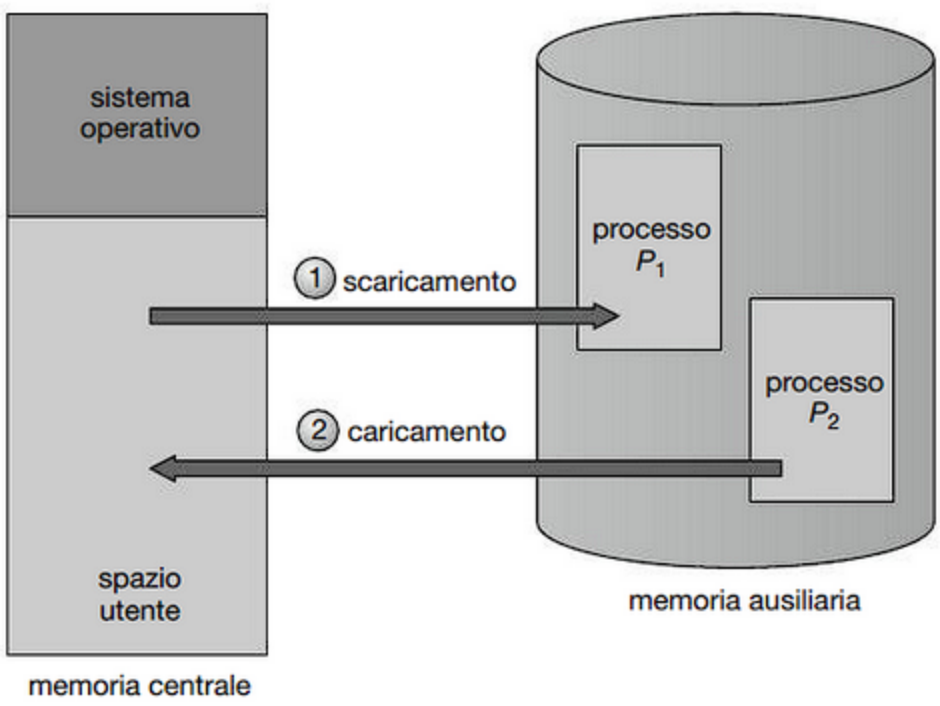
@redyz13 e @rosacarota

Si aumenta così lo spazio totale degli indirizzi fisici di tutti i processi, anche eccedendo la memoria fisica effettiva

La **backing store** è un disco veloce, grande abbastanza da memorizzare le copie delle immagini di memoria per tutti i processi (utenti) in esecuzione nel sistema

Deve poter provvedere accesso diretto a queste immagini di memoria

Swapping:



@redyz13 e @rosacarota

Lo swap in e lo swap out (caricamento e scaricamento) causano overhead

Una variante di questa politica di swapping viene utilizzata per gli algoritmi di scheduling basati su priorità

Quando si presenta la necessità di eseguire un processo con priorità maggiore, il gestore della memoria (se non c'è spazio in memoria centrale) effettua lo swap out di un altro con priorità inferiore, che verrà ripreso in memoria centrale appena quello con priorità maggiore termina

Questo tipo di swapping a volte viene detto **roll in, roll out**

Normalmente, un processo che è stato scaricato dalla memoria si deve ricaricare nello spazio di memoria occupato in precedenza. Questa limitazione è dovuta dal metodo di associazione degli indirizzi:

- Se l’associazione degli indirizzi logici agli indirizzi fisici della memoria (binding) viene effettuata al momento dell’assemblaggio o del caricamento allora il processo viene ricaricato nello stesso spazio di memoria occupato precedentemente
- Se il binding viene fatto a tempo di esecuzione allora il processo potrebbe essere caricato in uno spazio di memoria diverso

Il sistema mantiene una coda dei processi pronti (ready-queue) formata dai processi pronti per l’esecuzione, le cui immagini si trovano in memoria ausiliaria o in memoria centrale

Quando lo scheduler decide di eseguire un processo, chiama il dispatcher, che controlla se il primo processo della coda si trova in memoria centrale

Se il processo non è in memoria e non c’è spazio libero, allora il dispatcher esegue lo **swap out** di un processo presente in memoria e lo **swap in** del processo richiesto dallo scheduler della CPU
Quindi vengono caricati i registri e viene trasferito il controllo al processo che deve essere eseguito

Per utilizzare efficacemente la CPU è necessario che il tempo di esecuzione di ogni processo sia lungo rispetto al tempo di swap

La maggior parte del tempo di swap è dato dal tempo di trasferimento, che è direttamente proporzionale alla quantità di memoria sottoposta a swap

Lo swapping presenta diversi svantaggi:

- Sarebbe utile sapere quanta memoria viene effettivamente utilizzata per effettuare lo swap out riducendo così il tempo
- Quindi il sistema dovrebbe essere informato di quanta memoria un processo necessita tramite una system call effettuata dal

processo

- Per sottoporre il processo a swap è necessario che il processo sia inattivo. Particolare importanza si deve dare ai processi I/O pendente: mentre si vuole scaricare un processo per liberare memoria, tale processo può essere nell'attesa del completamento di un'operazione di I/O. Tuttavia, se un dispositivo di I/O accede in modo asincrono alle aree di I/O della memoria (buffer) d'utente, il processo non può essere scaricato
  - Es. Se un processo P_2 s'avvicendasse ad un processo P_1, l'operazione di I/O potrebbe tentare di usare la memoria che attualmente appartiene al processo P_2

  Questo problema si può risolvere in due modi:

  - Non scaricando dalla memoria un processo con operazioni di I/O pendenti
  - Eseguendo operazioni di I/O solo in buffer del sistema operativo in modo tale che i trasferimenti fra queste aree del SO e la memoria assegnata al processo possa avvenire solo quando il processo si trova in memoria centrale
    Questo meccanismo di memorizzazione (**double buffering**) aggiunge overhead

@redvz13 e @rosacarota

Attualmente lo swapping "semplice" è usato in pochi sistemi, richiede infatti un elevato tempo di gestione

Versioni modificate di swapping si trovano su molti sistemi, quali UNIX, Linux e Windows

Allocazione della memoria

La memoria centrale si divide di solito in due partizioni:

- **Sistema operativo residente**, generalmente mantenuto nella parte bassa della memoria, insieme al vettore degli interrupt

- **Processi utenti**, generalmente mantenuti nella parte alta della memoria

Per quanto riguarda la memoria, la protezione del S.O. dai processi utenti e la protezione dei processi utenti dagli altri processi utenti si può realizzare usando un registro di rilocazione insieme con un registro limite

Il registro di rilocazione contiene il valore dell'indirizzo fisico minore

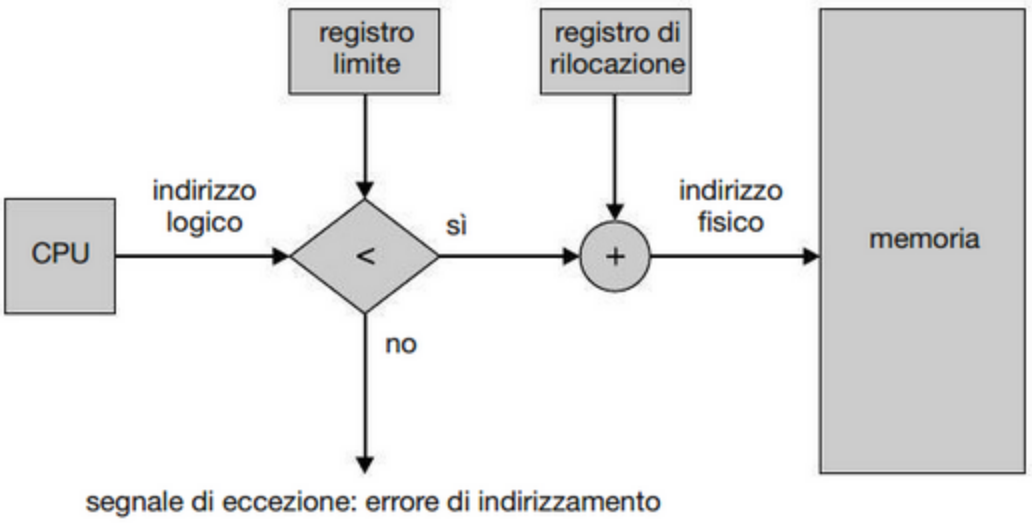
Il registro limite contiene il range di indirizzi logici:

- Ciascun indirizzo logico deve essere minore del contenuto del registro limite

La MMU fa corrispondere **dinamicamente** l'indirizzo fisico all'indirizzo logico sommando a quest'ultimo il valore contenuto nel registro di rilocazione

@redyz13 e @rosacarota

Registri di rilocazione e di limite:



Quando lo scheduler della CPU seleziona un processo per l'esecuzione, il dispatcher, durante il cambio di contesto, carica il registro di rilocazione e il registro limite con i valori corretti

La protezione si ha tramite il confronto di ogni indirizzo generato dalla CPU con il valore di questi registri

Grazie allo schema con registro di rilocazione, il S.O. ha dimensioni dinamiche: ad es. se un driver o un altro servizio del S.O. non è comunemente usato, risulta inutile tenere il suo codice e i suoi dati in memoria.
Questi tipi di codici vengono chiamati codice **transiente**: si inserisce a seconda delle necessità; l'utilizzo di questi codici cambia le dimensioni del S.O. durante l'esecuzione del programma

Allocazione contigua della memoria

@redyz13 e @rosacarota

Uno dei metodi più semplici di gestione della memoria consiste nel dividerla in partizioni di dimensione fissa (variabile?)

Ogni partizione deve contenere esattamente un processo, quindi il grado di multiprogrammazione è limitato dal numero di partizioni

Questo viene chiamato **metodo delle partizioni multiple:** quando una partizione è libera può essere occupata da un processo presene nella coda d'ingresso; terminato il processo, la partizione diventa di nuovo disponibile

Nello schema a partizione variabile, il S.O. conserva una tabella con le partizioni di memoria disponibili e quelle occupate

Inizialmente tutta la memoria è a disposizione dei processi utenti, cioè si tratta di un grande blocco di memoria disponibile: **buco**

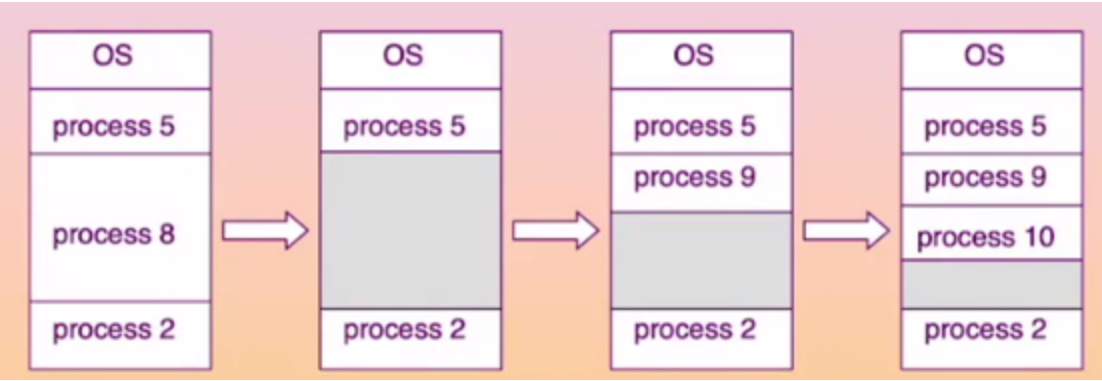
Buco - blocco di memoria disponibile

- Il sistema operativo deve mantenere informazioni in una tabella circa le partizioni allocate e quelle libere (buchi)
  La memoria si assegna ai processi della coda finché si possono soddisfare i requisiti di memoria del processo successivo, cioè finché esiste un buco disponibile sufficientemente grande ad accogliere quel processo
  Se ciò non accade, il S.O. può attendere finché non c'è un buco disponibile, o può scorrere la coda per soddisfare la richiesta di un altro processo

  Buchi di dimensioni diverse sono in generale presenti in memoria.

Quando un processo viene caricato in memoria, gli viene assegnata la memoria di un buco sufficientemente grande da contenerlo. Se il buco trovato è più grande, esso viene diviso e la restante parte torna nell'insieme dei buchi. Quando il processo termina, rilascia il blocco di memoria che aveva occupato che si reinserisce nell'insieme dei buchi: se si trova affianco ad altri buchi, si uniscono tutti i buchi adiacenti per formarne uno più grande

Questa procedura fa parte del problema più generale dell'**allocazione dinamica della memoria**: consiste nel soddisfare una richiesta di dimensione n data una lista di buchi liberi



Allocazione dinamica della memoria

Come si può soddisfare una richiesta di n blocchi di memoria?

- **First-fit:** allocare il primo buco abbastanza grande; la ricerca inizia sia dall'inizio dell'insieme di buchi sia dal punto in cui era terminata la ricerca precedente
- **Best-fit:** allocare il più piccolo buco abbastanza grande da contenere il processo; si deve compiere la sua ricerca in tutta la lista dei buchi liberi. Produce il più piccolo buco residuo (le parti di buco inutilizzate più piccole)
- **Worst-fit:** allocare il più grande buco disponibile; si deve compiere la sua ricerca in tutta la lista dei buchi liberi. Produce il più grande buco residuo

First-fit e best-fit sono più efficienti di worst-fit in termini di velocità e utilizzo della memoria

@redyz13 e @rosacarota

Frammentazione

Si frammenta lo spazio libero della memoria in tante piccole parti, tanti piccoli buchi

Frammentazione esterna:

- Lo spazio complessivo di memoria disponibile necessario per soddisfare la richiesta esiste, ma non è contiguo (first-fit e best-fit ne soffrono). La sua gravità dipende dalla quantità di memoria e dalla dimensione media dei processi
  Per l'algoritmo first-fit, per esempio, per n blocchi assegnati, se ne perdono $0,5\,n$ blocchi a causa della frammentazione: questa è nota come la **regola del 50 per cento**

Frammentazione interna:

- La memoria allocata è leggermente più grande di quella richiesta; questa memoria residua, interna alla partizione, non viene utilizzata

La frammentazione esterna può essere ridotta con un'operazione di **compattazione**

- Il contenuto della memoria viene riordinato per riunire la memoria libera in un solo grosso blocco
- Il compattamento è possibile solo se la rilocazione è dinamica e si compie in fase di esecuzione
- Un'altra possibile soluzione del problema è data dal consentire la non contiguità dello spazio degli indirizzi logici di un processo, permettendo così di assegnare la memoria fisica ai processi dovunque essa sia disponibile (si raggiunge tramite la segmentazione e la paginazione)
- Problemi relativi all'I/O:
  - Un job non può essere spostato in un'altra porzione di memoria mentre esegue operazioni di I/O
  - Possibile soluzione: fare I/O solo usando buffer nella porzione di memoria del S.O.

@redyz13 e @rosacarota

Segmentazione

La **segmentazione** è uno schema di gestione della memoria che imita la visione che il programmatore ha della memoria

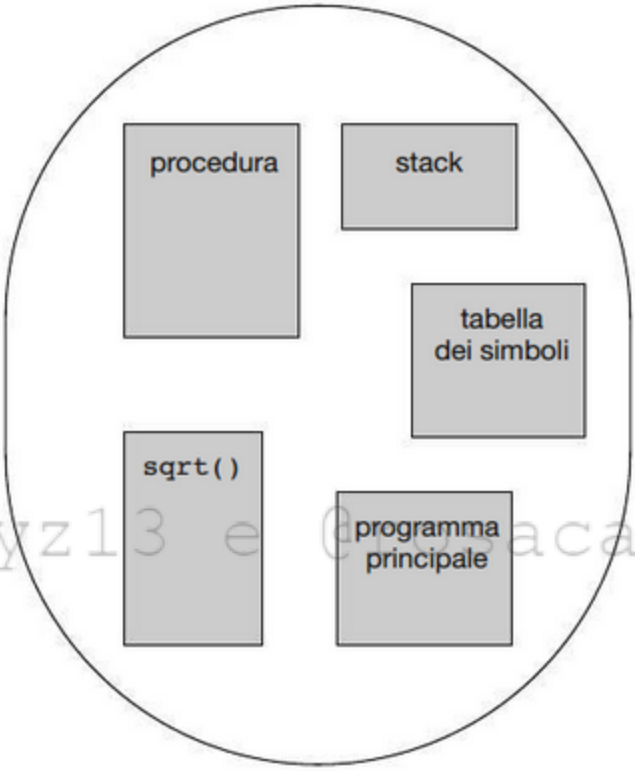
Un programma è costituito da un insieme di **segmenti**

Un segmento è un'unità logica, come:

- Programma principale
- Procedura, funzione
- Oggetto
- Variabili locali, variabili globali
- Blocchi comuni
- Stack

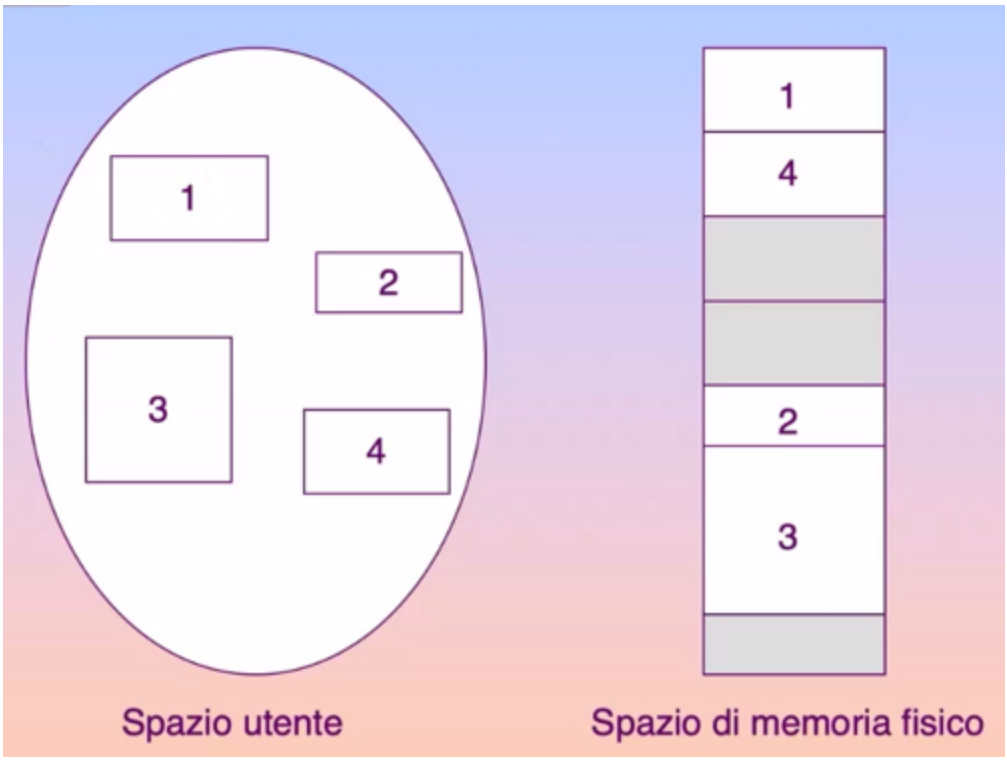
- Symbol table, array
- …

Un programma dal punto di vista del programmatore:



@redyz13 e @nosacarota

Visione logica della segmentazione:



@redyz13 e @rosacarota

Architettura di segmentazione

Uno spazio di indirizzi logici è composto da vari segmenti, ciascuno dei quali ha un nome e una lunghezza
Gli indirizzi specificano sia il nome sia l'offset all'interno del segmento, quindi il programmatore fornisce ogni indirizzo come una coppia ordinata di valori: un nome di segmento e un offset

Quindi, l'indirizzo logico consiste di una coppia:

- **<numero-di-segmento, scostamento>**

(per semplicità i segmenti sono numerati)

Occorre tradurre gli indirizzi logici in quelli fisici. Questo si fa tramite una tabella dei segmenti

Tabella dei segmenti - mappa, in uno spazio ad una dimensione (memoria fisica), gli indirizzi logici

Ogni elemento della tabella è una coppia ordinata:

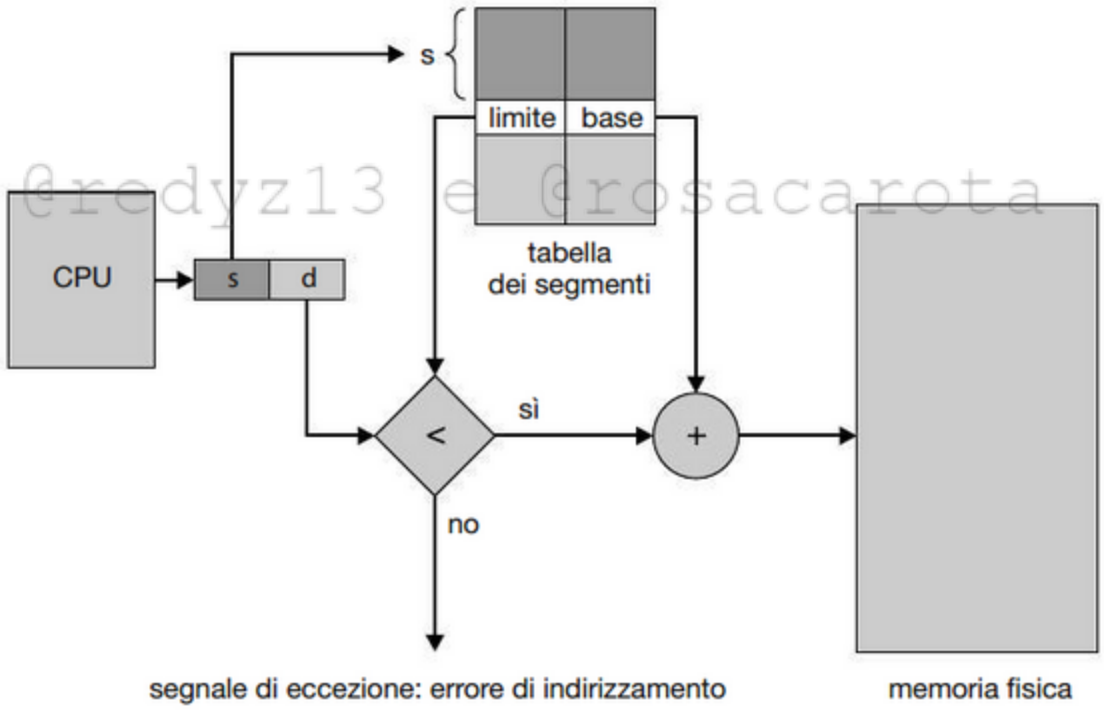
- **Base (del segmento)** - contiene l’indirizzo fisico iniziale della memoria nel quale il segmento risiede
- **Limite (del segmento)** - contiene la lunghezza del segmento

Segment table base register (STBR) - punta alla locazione in memoria della tabella dei segmenti

Segment table length register (STLR) - indica il numero di segmenti di un processo

- Il segmento numero s sarà legale se $s < STLR$

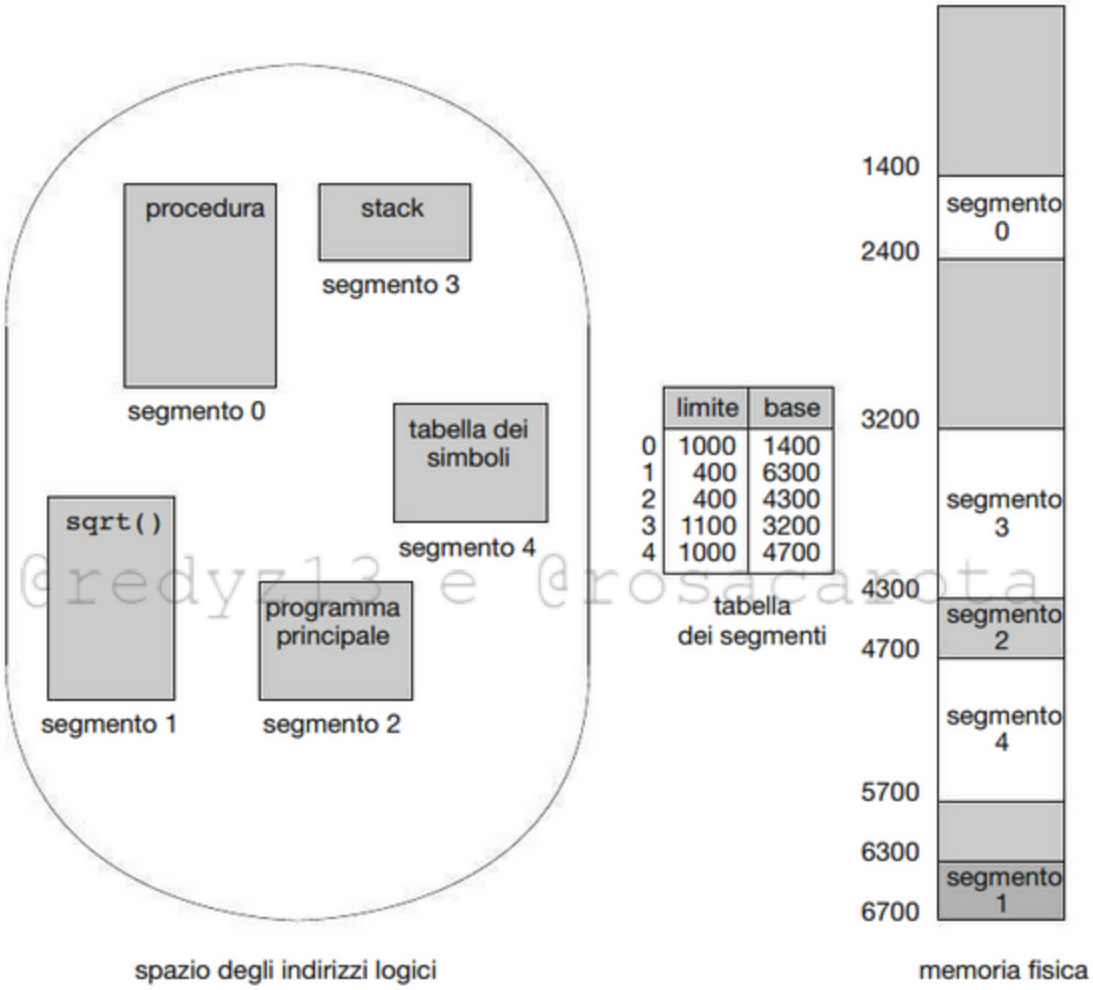
Architettura di segmentazione (schema):



Il numero del segmento si utilizza come indice della tabella dei segmenti; l’offset deve essere compreso tra 0 e il limite del segmento, altrimenti si genera un’eccezione per il sistema operativo. Se questa condizione sull’offset è soddisfatta,

questo viene sommato alla base del segmento per produrre l’indirizzo della memoria fisica

Esempio di segmentazione:



@redy713 e @rosacarota