

## Appunti di Sistemi Operativi e Reti

Domande e risposte - Parte di Sistemi Operativi (OS)

Capitolo 5 - Interfaccia e Struttura Kernel

1) Implementazione API attraverso chiamata di sistema

L'implementazione **API** mediante **system call** permette al sistema operativo di proteggere sé stesso dai programmi in esecuzione. La **CPU** può funzionare in modalità utente (**user mode**) o in modalità di sistema (**kernel mode**) impostando un opportuno bit di modalità. Alcune istruzioni sono privilegiate, ossia eseguibili solo in modalità di sistema e in user mode la **CPU** non può accedere alla **memoria del kernel**.

2) Cosa succede quando invocata un'istruzione macchina?

Viene generata un'**eccezione software**, ovvero la **CPU** passa in modalità di sistema e trasferisce il controllo ad una subroutine ad un determinato indirizzo di memoria. La **subroutine (system call interface)** legge il numero identificativo della chiamata di sistema, effettua un **lookup** da una tabella interna dell'indirizzo della routine che effettivamente implementa la chiamata di sistema, e salta a tale **indirizzo**. La routine invocata legge i parametri ed esegue la funzionalità richiesta e quando termina il processore passa in modalità utente.

3) Come avviene la transizione della **CPU** dalla modalità utente alla modalità di sistema?

Dal momento che l'invocazione delle chiamate di sistema passa per un'eccezione software, il passaggio di parametri risulta essere più complesso rispetto a quello di una normale chiamata di procedura. Vengono adottate tre metodologie:

1 - **Passaggio dei parametri nei registri del processore**: il vantaggio di questa metodologia è la rapidità mentre lo svantaggio è l'utilità solo per pochi parametri i cui tipi di dati hanno dimensione limitata.

2 - **Passaggio dell'indirizzo di memoria in uno dei registri ad un blocco nel quale vengono memorizzati i parametri**: usato da Linux in combinazione con il primo metodo.

3 - **Push dei parametri sullo stack**: il vantaggio di questa metodologia è la flessibilità (simile ad una procedure call), mentre lo svantaggio è la lentezza.

#### 4) Per quale motivo vengono utilizzati due stack

Ogni **thread** possiede solitamente due **stack**:

- il primo viene utilizzato dal programma in modalità utente;
- il secondo viene utilizzato quando il thread passa in modalità di sistema.

Una **chiamata di sistema**, come prima cosa, imposta lo **stack** del **thread** corrente allo stack di sistema, e al termine della chiamata di sistema ripristina lo **stack** a quello utente. I due **stack** vengono utilizzati per motivi di sicurezza perché dal momento che il processo potrebbe modificare a suo piacimento il **registro stack pointer** (che non è privilegiato) non si ha la certezza che questo punti ad uno stack "sano": pertanto in modalità di sistema occorre utilizzare uno stack "corretto".

#### 5) Utilizzo delle librerie dinamiche per le API

L'utilizzo delle librerie dinamiche per le **API** permette, in caso di aggiornamento del sistema operativo, non ricompilare/linkare le applicazioni qualora siano cambiate le chiamate di sistema, o l'implementazione delle **API**, purché l'interfaccia delle **API** rimanga equivalente. Il **vantaggio** di questa metodologia è l'implementazione delle **API** e delle **librerie standard del linguaggio** come **librerie dinamiche**. Se queste sono modificate (nell'implementazione, non nell'interfaccia), non occorre ricompilare tutti gli eseguibili per aggiornarli alla nuova versione delle **librerie**.

#### 5) ABI: Application Binary Interface

Si dice **application binary interface (ABI)**, l'insieme di convenzioni attraverso le quali il codice binario dell'applicazione si interfaccia con il codice delle librerie dinamiche delle **API**.

Un **eseguibile binario** può essere:

- portabile
- runtime portabile
- compilato → serve un eseguibile per ogni **sistema operativo**.

#### 6) Come è possibile avere applicazioni portabili su diversi sistemi di elaborazione?

Mediane tre possibili approcci:

- 1 - scrivere l'applicazione in un linguaggio con un interprete portatile (es. **Python, Ruby**): l'eseguibile in tal caso è il **sorgente**;
- 2 - scrivere l'applicazione in un linguaggio con un ambiente runtime portabile (es. **Java, .NET**): l'eseguibile in tal caso è il **bytecode**;

3 - scrivere l'applicazione utilizzando un linguaggio con un compilatore portabile ed API standardizzate: l'eseguibile è il file binario compilato e linkato.

Nei primi due casi l'eseguibile è normalmente uno solo per tutte le architetture, mentre nel terzo caso invece occorre, di norma, generare un eseguibile distinto a variazioni anche minime del sistema di elaborazione (spesso anche solo al variare della versione del sistema operativo). Il motivo di ciò può essere la differenza nell'architettura hardware: ad esempio, un file binario prodotto per **CPU ARM** non può essere interpretato da un sistema di elaborazione con **CPU x86-64**, dal momento che le istruzioni macchina delle due **CPU** differiscono.

### 7) Quali sono i sottosistemi del kernel?

I **sottosistemi del kernel** sono basati sulle categorie dei servizi offerti dal **kernel** stesso. I principali sono:

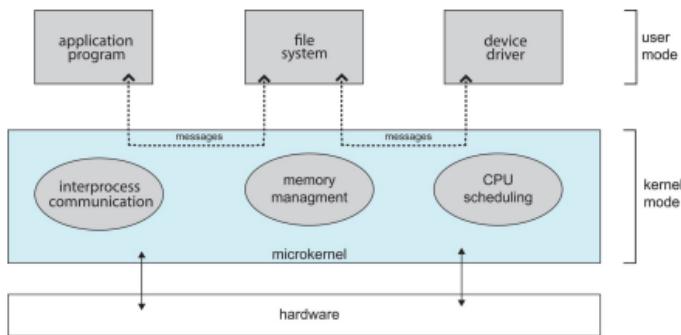
- gestione dei processi e dei thread;
- comunicazione tra processi e sincronizzazione;
- gestione della memoria;
- gestione dell'I/O;
- file system.

### 8) Che cosa si intende per organizzazione?

L'**organizzazione** è un programma complesso e di dimensioni elevate, che deve operare rapidamente e il cui malfunzionamento può provocare crash di sistema.

Esistono diversi modelli per progettarlo:

- **Monolitica**: singolo file binario statico.
  - 😊 : elevate prestazioni;
  - 😢 : complesso, fragile e non modulare.
- **A strati**: diviso in un insieme di livelli, lo strato più basso interagisce con l'hardware, e ogni strato interagisce con il successivo.
  - 😊 : indipendenza degli strati, astrazione delle caratteristiche della macchina;
  - 😢 : overhead.
- **Microkernel**: sposta quanti più servizi possibili fuori dal kernel, ha quindi dimensioni ridotte.
  - 😊 : estensibile, affidabile e meno sensibile ai crash;
  - 😢 : overhead, una richiesta deve transitare diversi "strati".



→ **A moduli**: strutturato in componenti dinamicamente caricabili (moduli), che parlano tra di loro attraverso interfacce: il kernel carica dinamicamente i moduli, solo quando offre il servizio implementato dal modulo, e lo scarica quando non serve più.

😊: tra strati e microkernel, con minore overhead e isolamento.

→ **Ibridi**: combinano diversi approcci allo scopo di ottenere sistemi indirizzati a una specifica prestazione.

### 9) Differenza tra politica e meccanismo

La **politica** dice quando una determinata operazione viene effettuata. Impatta profondamente sulle caratteristiche percepite del sistema di elaborazione. Il **meccanismo** spiega come una certa operazione è effettuata. Esso è più stabile delle politiche, che spesso cambiano in funzione delle caratteristiche che il sistema di elaborazione deve avere.

### Capitolo 6 - Processi e Thread: Struttura (Scheduling della CPU)

#### 10) Che ruolo ha lo scheduler della CPU?

Lo **scheduler della CPU**, o **scheduler a breve termine**, seleziona un **processo** tra quelli nella **ready queue** ed alloca un core libero ad esso. Tali assegnamenti possono essere effettuati in diversi momenti, corrispondenti a cambi di stato dei processi che possono essere:

- quando un processo passa da stato running a stato waiting;
- quando un processo passa da stato running a stato ready;
- quando un processo passa da stato waiting a stato ready;
- quando un processo termina.

#### 11) Differenza tra schema di scheduling preemptive e non preemptive?

Lo **schema di scheduling** viene definito **non preemptive** se il riassegnamento viene effettuato quando un processo passa da stato running a stato waiting oppure quando un processo termina. Tale schema può essere definito anche **cooperativo**. Lo **schema di scheduling** viene definito **preemptive** se il

riassegnamento quando un processo passa da stato running a stato ready oppure quando un processo passa da stato waiting a stato ready.

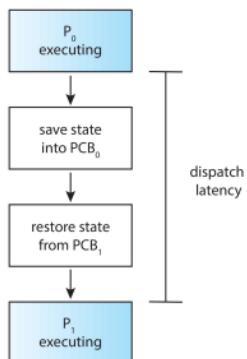
### 12) Qual è il ruolo dello dispatcher?

Il **dispatcher** passa effettivamente il **controllo della CPU** al **processo scelto** dallo **scheduler** a breve termine svolgendo i seguenti passi:

- effettua il cambio di contesto;
- passa in modalità utente;
- salta nel punto corretto del programma del processo selezionato (ossia, dove era stato precedentemente interrotto).

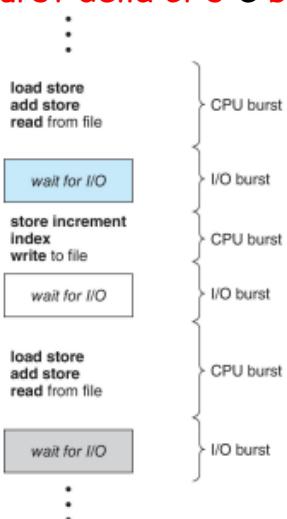
### 13) Cosa significa latenza di dispatch?

Si definisce **latenza di dispatch** il tempo impiegato dal dispatcher per fermare un processo ed avviare un altro. Lo scheduler implementa una politica mentre il dispatcher implementa un meccanismo.



### 14) Burst CPU e Burst I/O: definizioni

Si definisce **burst di CPU** una sequenza di operazioni della **CPU**, mentre **burst I/O** è l'attesa completamento operazione di **I/O**. Gli **algoritmi di scheduling** sfruttano il fatto che di norma l'esecuzione di un processo è una sequenza di **burst della CPU e burst dell'I/O**.



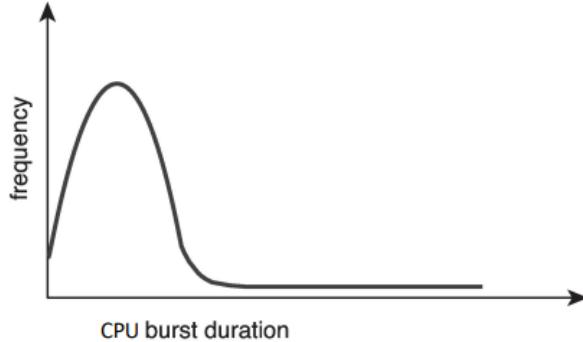
### 15) Distribuzione delle durate dei burst della CPU: caratteristiche

Le due possibili casistiche sono:

→ **programma con prevalenza di I/O (I/O bound)**: elevato numero di **burst CPU** brevi, numero ridotto di **burst CPU** lunghi (tipico dei programmi interattivi);

→ **programma con prevalenza di CPU (CPU bound)**: elevato numero di **burst CPU** lunghi, numero ridotto di **burst CPU** brevi (tipico dei programmi interattivi).

In entrambi i casi la curva della distribuzione ha la seguente forma:



### 16) Criteri di scheduling: caratteristiche

I criteri di **scheduling** sono misure che servono per confrontare le caratteristiche dei diversi algoritmi che non dipendono solo dall'algoritmo, ma anche dal carico. I principali criteri:

→ **uso di CPU**: % di tempo in cui la CPU è attiva nell'esecuzione dei processi utente (dovrebbe essere tra il 40% e il 90%, in funzione del carico);

→ **throughput**: numero di processi che completano l'esecuzione nell'unità di tempo (dipende dalla durata dei processi);

→ **tempo di completamento**: tempo necessario per completare l'esecuzione di un certo **processo** (dipende da molti fattori: durata del processo, carico totale, durata dell'I/O...);

→ **tempo di attesa**: tempo trascorso dal processo nella **ready queue** (meglio del tempo di completamento, meno dipendente da durata del processo e dell'I/O)

→ **tempo di risposta**: negli ambienti interattivi, tempo trascorso tra l'arrivo di una richiesta al processo e la produzione della prima risposta, senza l'emissione di questa nell'output.

### 17) Algoritmo: Scheduling ordine di arrivo - FCFS (First Come First Served)

La **CPU** viene assegnata al primo processo che la richiede e comporta un'implementazione molto semplice (**vantaggio**) ma un tempo di attesa medio può essere lungo (**svantaggio**).

**Esempio:**

Processo	Durata burst CPU	Tempo attesa
P <sub>1</sub>	24	0
P <sub>2</sub>	3	24
P <sub>3</sub>	3	27



$$T_w = (0 + 24 + 27) / 3 = 17$$

$$T_{completamento} = (24 + 27 + 30) / 3 = 27$$

### 18) Algoritmo: Scheduling per brevità - SJF (Shortest Job First)

La **CPU** viene assegnata al processo che ha il successivo **CPU burst** più breve e comporta un'implementazione quasi identica a **FCFS**, ma minimizza il tempo di attesa medio (è ottimale - **vantaggio**) ma di solito non si sa in anticipo qual è il processo che avrà il **CPU burst** più breve (**svantaggio**).

Processo	Durata burst CPU	Tempo attesa
P <sub>1</sub>	6	0
P <sub>2</sub>	8	6
P <sub>3</sub>	7	14
P <sub>4</sub>	3	21

Applicando **SJF** si ottiene quindi:

Processo	Durata burst CPU	Tempo attesa
P <sub>4</sub>	3	0
P <sub>1</sub>	6	3
P <sub>3</sub>	7	9
P <sub>2</sub>	8	16

$$T_w = (0 + 3 + 9 + 16) / 4 = 7$$

$$T_{completamento} = (3 + 9 + 16 + 24) / 4 = 13$$

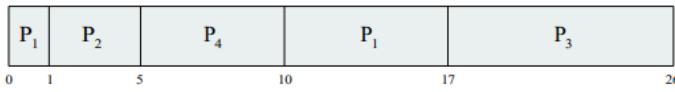
### 19) Algoritmo: Scheduling per brevità - SRTF (Shortest Remaining Time First)

L'algoritmo **shortest-remaining-time-first** (SRTF) utilizza la prelazione per

gestire il caso in cui i processi non arrivino tutti nello stesso istante: se nella ready queue arriva un processo con un burst più corto di quello running, quest'ultimo viene prelazionato dal nuovo processo.

Processo	Tempo di arrivo	Durata burst CPU
P <sub>1</sub>	0	8
P <sub>2</sub>	1	4
P <sub>3</sub>	2	9
P <sub>4</sub>	3	5

Determina l'ordine di arrivo



$$\text{Tempo di attesa medio} = ((17-8) + (5-5) + (26-11) + (10-8)) / 4 = 6,5$$

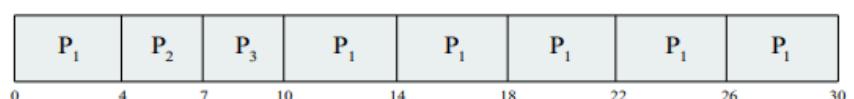
$$\text{Tempo di completamento medio} = ((17-0) + (5-1) + (26-2) + (10-3)) / 4 = 13$$

## 20) Algoritmo: Scheduling circolare - RR (Round Robin)

Nello **scheduling circolare**, o **round-robin (RR)**, ogni processo ottiene una piccola quantità fissata di tempo di **CPU** (quanto di tempo), di solito 10-100 millisecondi, per il quale può essere in esecuzione. Trascorso tale tempo il processo in esecuzione viene interrotto e messo in fondo alla **ready queue**, che è gestita in maniera **FIFO** e funziona essenzialmente come un buffer circolare in cui i processi vengono scanditi dal primo all'ultimo, per poi ripartire dal primo nello stesso ordine. Si dispone di un **timer** che genera un **interrupt** periodico con periodo **q** per effettuare la prelazione del processo corrente (**passaggio del processo da stato running a ready**).

Esempio

Processo	Durata burst CPU
P <sub>1</sub>	24
P <sub>2</sub>	3
P <sub>3</sub>	3



$$q = 4$$

$$\text{Tempo di attesa medio} = ((30-24) + (7-3) + (10-3)) / 3 = 5,67$$

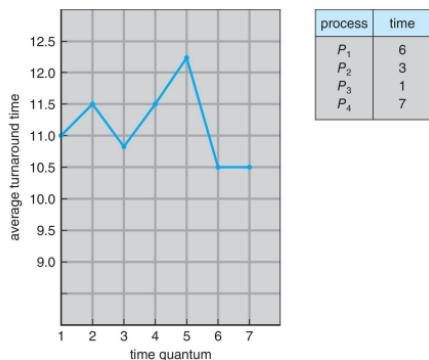
$$\text{Tempo di completamento medio} = (30 + 7 + 10) / 3 = 15,67$$

## Attenzione

Se ci sono **n** processi nella ready queue e il quanto temporale è **q**, allora nessun processo attende più di **q \* (n - 1)** unità di tempo nella ready queue prima di ridiventare running per un altro quanto di tempo. Se **q** è elevato, allora RR tende al FCFS, altrimenti questa si «mangia» un tempo comparabile al tempo di esecuzione dei processi utente e l'utilizzo della CPU diventa inaccettabilmente basso.

## Considerazione

Il **tempo di completamento medio** migliora se la maggioranza (~80%) dei **CPU burst** è più breve di **q**.



## 21) Algoritmo: Scheduling con priorità

Nello **scheduling con priorità** ad ogni processo è associato un numero intero che indica la sua priorità e viene eseguito il **processo con priorità più alta**, gli altri aspettano (Unix - Like numero più basso indica la priorità più alta, mentre in Windows il contrario). In tale **algoritmo** potrebbe verificarsi il problema della **attesa indefinita (starvation)**, in cui un processo a priorità troppo bassa potrebbe non venir mai schedulato e tale problema potrebbe essere risolto mediante l'**invecchiamento (aging)**, ossia aumento automatico di priorità di un **processo** al crescere del tempo di permanenza nella **ready queue**.

### Esempio 1

Processo	Durata burst CPU	Priorità (UNIX)
$P_1$	10	3
$P_2$	1	1
$P_3$	2	4
$P_4$	1	5
$P_5$	5	2



$$T_w = (0 + 1 + 6 + 16 + 18) / 5 = (23 + 18) / 5 = 41 / 5 = 8,2$$

$$T_{completamento} = (1 + 6 + 16 + 18 + 19) / 5 = (23 + 37) / 5 = 12$$

### Esempio 2

Processo	Durata burst CPU	Priorità (UNIX)
$P_1$	4	3
$P_2$	5	2
$P_3$	8	2
$P_4$	7	1
$P_5$	3	3



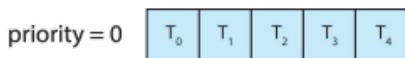
$q = 2$  per processi con stessa priorità

$$\text{Tempo di attesa medio} = ((26 - 4) + (16 - 5) + (20 - 8) + (7 - 7) + (27 - 3)) / 5 = 13,8$$

$$\text{Tempo di completamento medio} = (26 + 16 + 20 + 7 + 27) / 5 = 19,2$$

## 22) Introdurre il concetto di coda multilivello

La "coda multilivello" viene utilizzata per organizzare e schedulare i processi in modo efficiente suddividendoli in più code basate su alcune caratteristiche come la priorità, tempo di esecuzione, ecc.

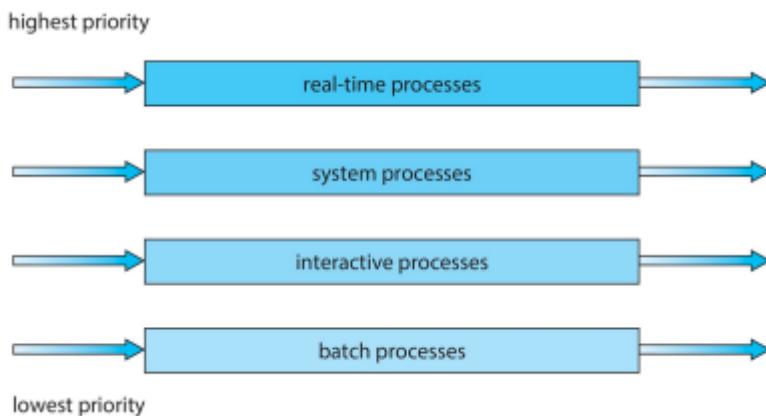


•  
•  
•



## 23) Introdurre il concetto di code multilivello con retroazione

Uno **scheduler** è definito con **code multilivello con retroazione** se la priorità di un processo può variare dinamicamente: è sufficiente spostarlo da una **ready queue** ad una certa priorità ad un'altra. L'invecchiamento è di solito implementato spostando un **processo** verso una **coda a priorità più alta**. Lo stesso approccio viene adottato anche per l'identificazione di un **processo** come **I/O bound** o **CPU bound** in modo da modificarne il **cambio di priorità**.



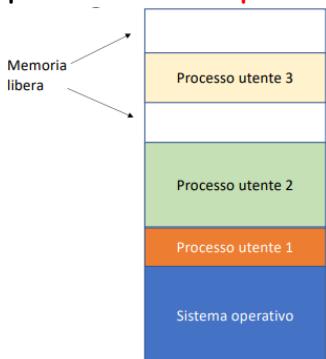
## Capitolo 7 - Gestione della memoria

### 24) Quali sono le caratteristiche di un sistema multiprogrammato?

In un **sistema multiprogrammato**, più **immagini** di più **processi** sono contemporaneamente nella **memoria centrale**. Il **sistema operativo** deve, pertanto, allocare porzioni di memoria centrale ai diversi **processi** in funzione delle necessità di tali **processi**.

## 25) Descrivere la strategia di allocazione contigua

La **strategia di allocazione contigua** è l'approccio più semplice di allocazione della **memoria** in un **sistema multiprogrammato**, in cui 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.

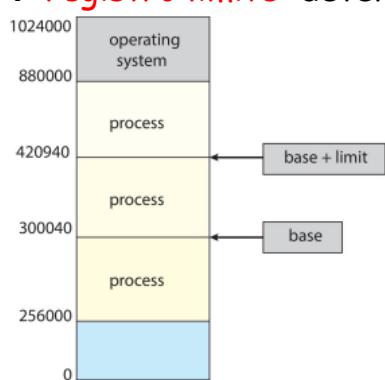


## 26) Descrivere la protezione con registro base e limite

La **protezione con registro base e limite** è il più semplice metodo di protezione utilizzabile con l'**allocazione contigua**. Il processore possiede due registri:

→ **registro base**: contiene il più piccolo indirizzo della memoria fisica che il processo corrente ha il permesso di accedere;

→ **registro limite**: determina la dimensione dell'intervallo ammesso.



## 27) Caratteristiche della protezione con registro base e limite

Le caratteristiche sono:

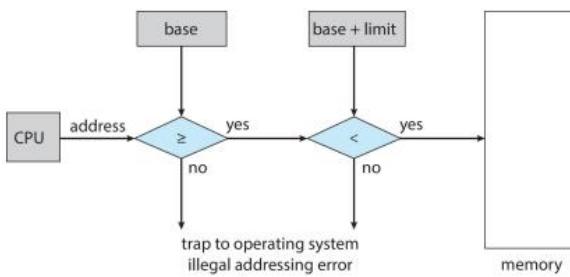
→ i registri base e limite possono essere impostati solo in modalità di sistema;

→ 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**;

→ in modalità di sistema il **processore** può accedere a tutta la **memoria**.



### 28) Quali sono i svantaggi della soluzione con registro base e limite?

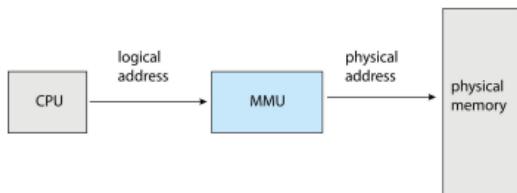
Lo svantaggio con della soluzione con registro base e limite è che non viene fornito uno spazio di indirizzamento virtuale ai processi e l'unico modo per implementare uno spazio di indirizzamento virtuale con tale soluzione sarebbe il binding in fase di esecuzione.

### 29) Differenza tra indirizzo logico e fisico

Si definisce **indirizzo logico** un indirizzo generato dalla **CPU** durante l'esecuzione di un programma, mentre l'**indirizzo fisico** è l'indirizzo appartenente alla **memoria centrale**.

### 30) Cos'è la MMU (Memory Management Unit)?

Si definisce **MMU (Memory Management Unit)** è il dispositivo hardware che traduce **indirizzi logici** in **indirizzi fisici**. Esso interviene solo in **modalità utente**: in **modalità kernel** gli indirizzi generati dalla **CPU** sono direttamente **indirizzi fisici**.

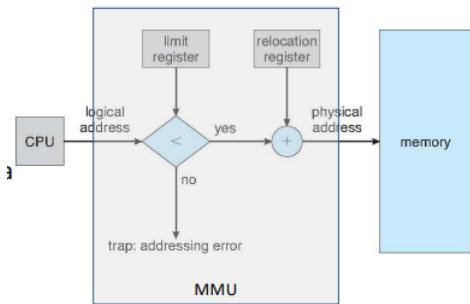


### 31) Caratteristiche di MMU

Le caratteristiche sono:

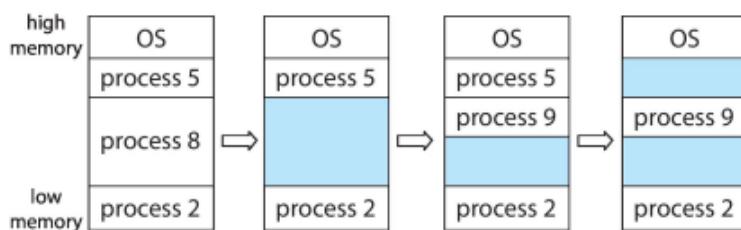
- la **MMU** più semplice utilizzabile con lo schema di **allocazione contigua** è la **MMU con registro di rilocazione**;
- risulta essere una variazione dello schema con **registri base e limite**, dove il **registro base** è ora chiamato **registro di rilocazione**;
- l'**indirizzo fisico** è ottenuto sommando all'**indirizzo logico** il valore del **registro di rilocazione** e in tal modo i programmi hanno l'illusione di avere uno **spazio di indirizzamento virtuale** che va dall'**indirizzo 0** a un indirizzo massimo pari al valore contenuto nel **registro limite**;
- 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**).



### 32) Definizione di schema a partizioni variabili e buco

Si definisce **schema a partizioni variabili** una 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.



### 33) Quali sono le tre tipologie di allocazione contigua a partizioni variabili?

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.

### 34) Differenza tra frammentazione interna ed esterna

Si definisce **frammentazione esterna**, la tipologia di frammentazione in cui la memoria libera è sufficiente per la creazione di un nuovo processo, ma è sparsa tra buchi non contigui troppo piccoli. Si definisce **frammentazione interna**, la tipologia di **frammentazione** in cui la memoria allocata ad un processo è più

grande della memoria necessaria, una **partizione** può contenere memoria inutilizzata.

### 35) Cosa consiste il processo di paginazione?

L'idea della **paginazione** è quella di permettere una allocazione di memoria ai processi non contigua e la **memoria centrale** viene divisa in **frames**, ossia blocchi di dimensione fissa (tra 512 bytes e 16 Mbytes). Similmente lo spazio di **indirizzamento virtuale** di ogni **processo** è diviso in **pagine**, ossia blocchi delle stesse dimensioni dei **frames**.

### 36) Che ruolo ha la tabella delle pagine?

Una **tabella delle pagine** associa le **pagine** di un processo ai **frames** in memoria centrale, e permette alla **MMU** di tradurre gli **indirizzi logici** in **fisici**.

### 37) Quali sono i vantaggi e svantaggi della paginazione?

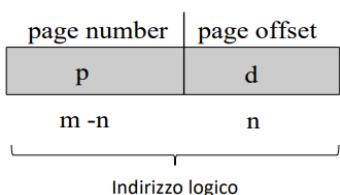
I **vantaggi** sono l'assenza di **frammentazione esterna** e vi è piena indipendenza tra **indirizzi logici** e **indirizzi fisici** (ad esempio, si possono avere **indirizzi logici** a **64 bit** anche se la **memoria fisica** è più piccola). Lo **svantaggio** è la presenza di **frammentazione interna**.

### 38) Da cosa è costituito un indirizzo logico?

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).

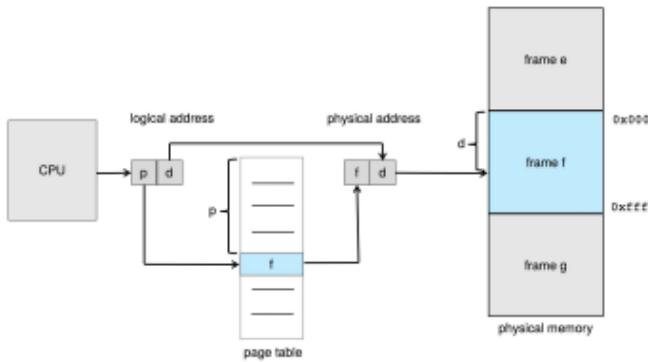
Se lo spazio di **indirizzi logici** ha dimensione  $2^m$  e le pagine hanno dimensione  $2^n$ , allora il numero totale di pagine è  $2^{m-n}$ .



### 39) Come viene tradotto un indirizzo logico in fisico?

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**.



40) Come avviene la frammentazione interna e quanto sono grandi le pagine nei vari sistemi operativi?

Nel caso medio la **frammentazione interna** è di mezzo **frame per processo** e questo suggerisce di fare pagine di dimensioni ridotte per ridurre lo spreco di memoria. Se aumenta il numero totale di pagine, di conseguenza aumenta la dimensione della tabella delle pagine e 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.

41) Che compito ha il sistema operativo nel supporto alla paginazione?

Il **sistema operativo** deve mantenere la **tabella delle pagine** di ciascun **processo** e una **tabella dei frame**, che indica lo stato di ogni **frame** (libero o assegnato, e in tal caso a che processo/i). Se il **kernel** deve accedere alla **memoria** di un **processo**, deve effettuare "manualmente" la traduzione degli **indirizzi logici** del **processo** in **indirizzi fisici** (quando la **CPU** è in modalità di sistema la **MMU** non è attiva).

42) Cosa utilizza la MMU per effettuare la paginazione?

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.

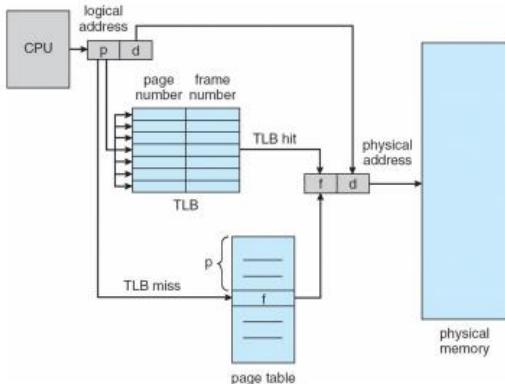
In tale schema ogni accesso a dati/istruzioni richiede due accessi in memoria, uno alla tabella delle pagine, ed uno per recuperare il dato. Per risolvere ciò si utilizza una cache apposita per la **tabella delle pagine** chiamata **translation lookaside buffer (TLB)**.

43) Quali sono le caratteristiche del Translation Lookaside Buffer (TLB)?

Le caratteristiche del **Translation Lookaside Buffer (TLB)**:

→ il **TLB** è di solito piccolo (da 64 a 1024 entries);

- ad ogni cambio di contesto devo effettuare il flush del **TLB**, il che comporta una notevole riduzione di prestazioni;
- per risolvere tale problema 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.



#### 44) Differenza tra TLB miss e TLB hit

Ho un **TLB hit** se il numero di pagina di un indirizzo logico si trova nel TLB, altrimenti ho un **TLB miss**.

#### 45) Come si calcola il tasso di successi?

Il **tasso di successi (hit ratio)** è dato dalla percentuale di **TLB hit** sul totale degli accessi.

#### 46) Come si calcola il tempo medio di accesso alla memoria?

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:

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

#### 47) Politiche di sostituzione nel TLB

Si hanno diverse strategie di sostituzione nel TLB:

→ LRU (Last Recently Used);

→ Round - Robin;

→ Casuale.

Alcuni processori generano un interrupt in maniera da permettere al sistema operativo di partecipare alla decisione di quale entry sostituire, mentre altri processori permettono al sistema operativo di vincolare (**wire down**) delle **TLB** entry che non vogliono siano mai sostituite.

#### 48) Caratteristiche del registro PTLR

Il **registro PTLR** permette di "tagliare" la tabella delle pagine in maniera da ridurre gli indirizzi logici disponibili al processo. Il **vantaggio** di tale **registro** è la possibilità di ridurre la dimensione della tabella delle pagine. Risulta possibile anche mettere in ogni **entry** della tabella delle pagine un **bit di validità**, che indica se l'**entry** è valida o no (ossia, se la pagina è effettivamente mappata su un **frame**).

#### 49) Vantaggio del bit di validità e quali sono altri bit contenuti nella tabella delle pagine?

Il **bit di validità** offre una maggiore flessibilità in quanto risulta possibile creare intervalli di **indirizzi logici** inaccessibili in qualsiasi punto dello spazio di **indirizzi logici**. Altri **bit** contenuti nella **tabella delle pagine** permettono di indicare se il **frame** associato è:

- **read-only** o **read-write** (**bit di protezione**);
- **eseguibile** o no (**bit di esecuzione**).

#### 50) Caratteristiche delle pagine condivise

La **paginazione** permette di condividere più facilmente **memoria fisica** tra più **processi**: è sufficiente che i **frame** siano nelle **tabelle delle pagine** dei **processi** che li condividono.

#### 51) Perché sono state introdotte delle strutture delle tabelle delle pagine per grandi spazi di indirizzamento virtuali?

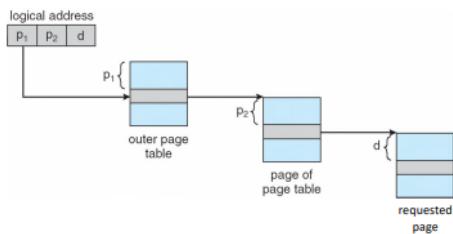
Dato che con **spazi di indirizzamento virtuali grandi** (32 o 64 bit) le **tabelle delle pagine** possono diventare a loro volta molto grandi, ossia se le pagine sono grandi  $2^n$  e lo spazio di indirizzamento virtuale  $2^m$ , le pagine sono in tutto  $2^{m-n}$  e supponendo che ogni entry **occupa** **e bytes** di dimensione, in totale la tabella delle pagine ha dimensione  $e * 2^{m-n}$ , sono state introdotte varie soluzioni per strutturare le **tabelle delle pagine** in maniera che siano sufficientemente piccole/efficienti.

#### 52) Quali sono le strutture delle tabelle delle pagine che supportano grandi spazi di indirizzamento virtuali?

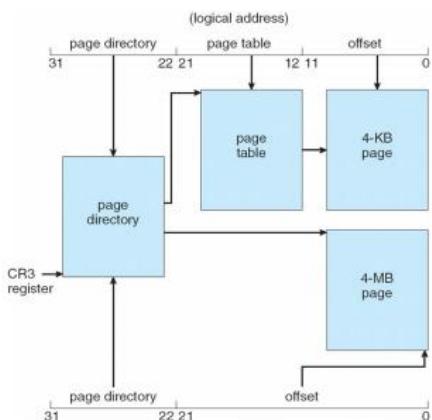
Le strutture sono:

- **Tabelle delle pagine gerarchiche**: servono per evitare l'allocazione di una **tabella delle pagine** in una regione di memoria contigua introducendo una **tabella a due livelli**, in cui ogni numero di pagina è diviso a sua volta in un numero di

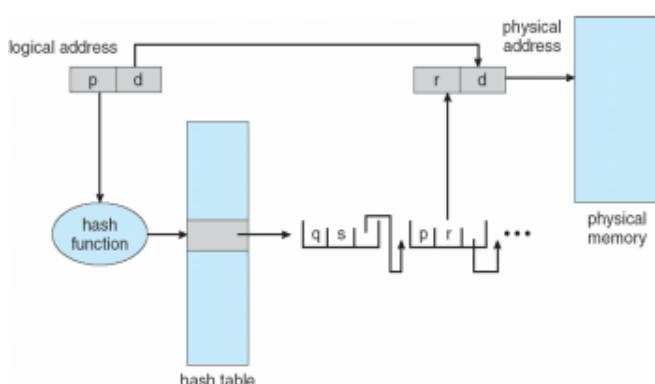
pagina e in un offset e i livelli vengono utilizzati per recuperare da una tabella esterna delle pagine l'indirizzo della pagina della tabella delle pagine costruendo infine l'**indirizzo fisico**.



Il **vantaggio** è che la **tabella delle pagine gerarchica** permette di supportare contemporaneamente pagine di dimensioni diverse. Viene effettuato ciò marcando un'entry nella tabella delle pagine esterne perché sia considerata un'entry di ultimo livello e ciò 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 l'aumento del numero di accessi in memoria per recuperare un dato/istruzione nel caso pessimo.



→ **Tabelle delle pagine di tipo hash:** la **tabella delle pagine** è organizzata come una **tabella hash** applicando una funzione hash (semplice!) al numero di pagina. Ogni entry della tabella ha una **lista concatenata** di elementi per gestire le eventuali collisioni.



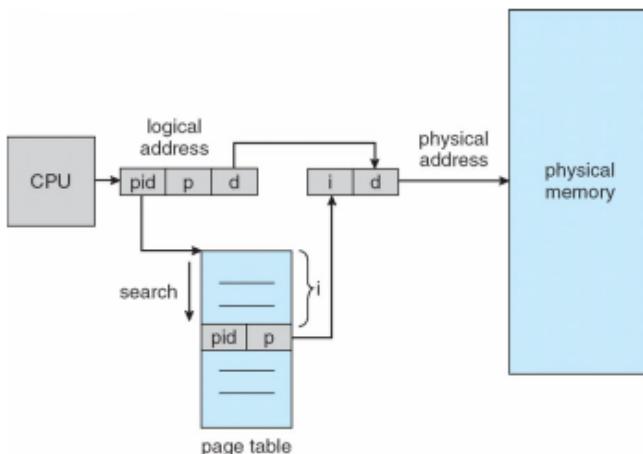
→ **Tabelle delle pagine invertite:** implementata introducendo un entry per ogni

frame, anziché per ogni pagina (vengono tracciati i **frame** anziché le pagine) ed ogni entry riporta il numero di pagina del corrispondente frame, più informazioni addizionali tra cui l'**ASID**. I **vantaggi** sono:

- 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.

Gli **svantaggi** sono:

- maggiore tempo di accesso (occorre cercare l'entry);
- non è possibile condividere pagine tra processi diversi.



### 53) Cosa si intende per swapping?

Lo **swapping** è una tecnica che permette di eseguire più processi di quanti la memoria fisica ne possa contenere e ciò viene effettuato spostando temporaneamente l'immagine di un processo pronto o in attesa dalla memoria centrale in una memoria secondaria (backing store, di solito il disco) sospendendo tale processo.

### 54) Qual'è lo scopo dello swapping?

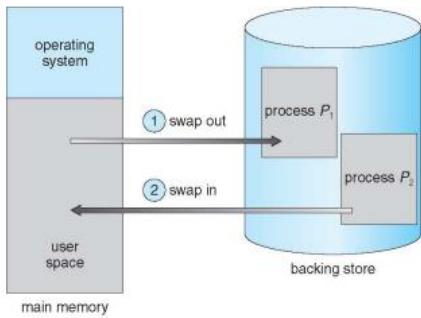
L'obiettivo dello **swapping** è permettere ad un altro processo di andare in memoria, e quindi in prospettiva di andare in esecuzione. Si hanno due varianti di swapping:

- **swapping standard**;
- **swapping con paginazione**.

### 55) Swapping standard: caratteristiche

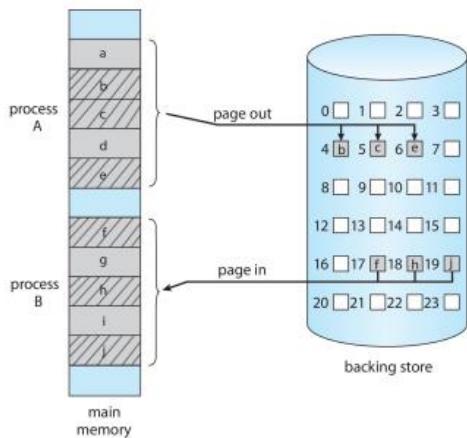
Nello **swapping standard** un intero processo viene spostato dalla memoria centrale alla **backing store (swap out)**. Occorre spostare anche tutte le strutture dati del sistema operativo relative al **processo** e ai **threads**. Lo **svantaggio** di tale approccio è che spostare un **intero processo** è molto oneroso

(solo Solaris usa ancora lo **swapping standard** in circostanze estreme).



## 56) Swapping con paginazione

Lo **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**. Risulta essere molto meno oneroso dello **swapping standard** e l'operazione di caricamento di una pagina dalla **memoria centrale** è detta **page out**, l'operazione inversa è detta **page in**.



## 57) Cosa si intende per grado di multiprogrammazione?

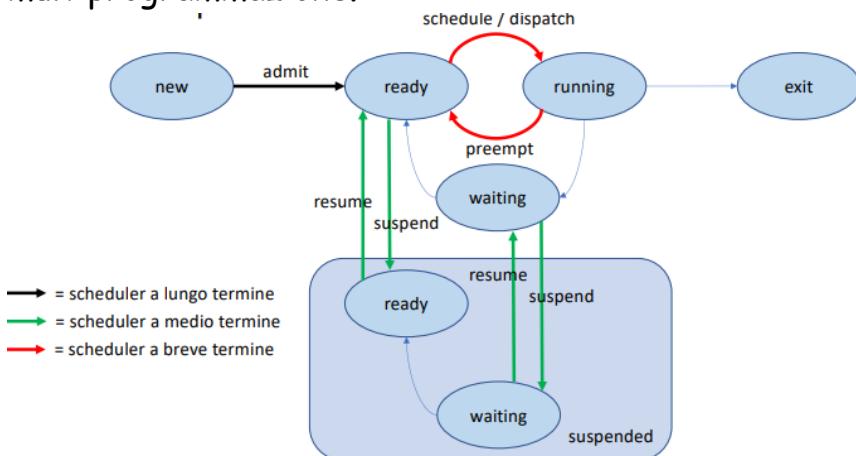
Si definisce **grado di multiprogrammazione** il numero massimo di **processi** che lo **scheduler** della **CPU** può mandare in esecuzione (ready + running + waiting). Il **grado di multiprogrammazione** determina l'occupazione totale di memoria fisica da parte dei **processi** e in assenza di memoria virtuale, un processo può essere mandato in esecuzione solo se la sua **immagine** è interamente in memoria. Il **grado di multiprogrammazione** è uguale al numero di **immagini in memoria** ed è anche correlato con l'utilizzazione delle **CPU**:

- se è basso allora anche l'utilizzazione tende ad essere bassa, perché quando i processi vanno in attesa non ci sono abbastanza processi ready per tenere occupati i processori;
- al crescere del grado di multiprogrammazione aumenta il numero di processi che, in media, sono ready, e quindi anche l'utilizzazione dei processori.

58) Quali sono le tre tipologie di scheduler che controllano il grado di multiprogrammazione?

Si hanno tre tipologie di scheduler:

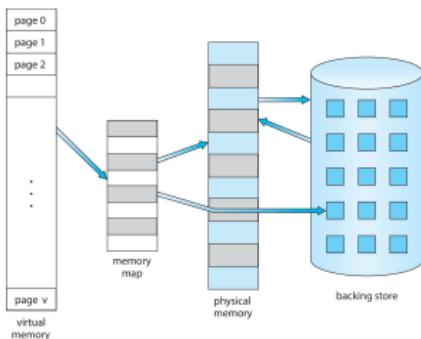
- **Scheduler a lungo termine (o admission scheduler)**: decide se ammettere un nuovo processo dopo la sua creazione nella ready queue: in questo modo influisce sul grado di multiprogrammazione sul lungo periodo.
- **Scheduler a medio termine**: sospende un processo per farne swapping e portarlo fuori memoria, e viceversa fa riprendere dalla sospensione un processo riportandolo in memoria: in questo modo influisce sul grado di multiprogrammazione sul medio periodo.
- **Scheduler a breve termine (CPU scheduler)**: non influisce sul grado di multiprogrammazione.



59) Cosa si intende per memoria virtuale?

La **memoria virtuale** è la completa separazione tra memoria logica e memoria fisica di un programma in cui un **processo** può essere eseguito anche se solo una parte di esso è in **memoria fisica**. Lo **spazio di indirizzamento virtuale** può essere molto più grande dello spazio di indirizzamento fisico e si hanno due possibili implementazioni:

- paginazione su richiesta;
- segmentazione su richiesta.



### 60) Caratteristiche della paginazione su richiesta

La **paginazione su richiesta** (**demand paging**) è basata sull'hardware di paginazione e l'idea di tale approccio è quella di non portare l'intero processo in memoria alla creazione, ma solo le pagine che vengono volta per volta usate e il resto del processo risiede nella **Backing store**. Simile allo **swapping** con **paginazione**, ma questo individua le pagine da scaricare/caricare in maniera predittiva, mentre la **paginazione** su richiesta lo fa in base all'uso.

### 61) Come viene implementata la paginazione su richiesta?

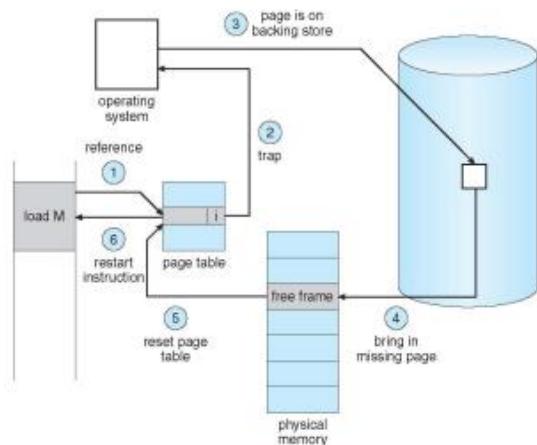
L'implementazione è fondata sui seguenti passaggi:

- l'**hardware di paginazione** deve fornire un opportuno supporto alla paginazione su richiesta;
- la **tabella delle pagine** deve possedere il **bit di validità** per indicare se una certa pagina è non valida oppure assente dalla memoria;
- una volta che il programma tenta un accesso ad una pagina il cui bit di validità è impostato a zero la **MMU** genera un'interruzione di **page fault**;
- il **sistema operativo** a questo punto gestisce il page fault, e se questo è stato generato perché la pagina è assente dalla memoria la porta in memoria;
- l'**hardware** deve inoltre permettere la riesecuzione dell'istruzione interrotta da un **page fault** in maniera trasparente al **programma**;
- infine occorre un'area opportuna (**swap space**) nella **Backing store** dove memorizzare le **pagine fuori memoria**.

### 62) Come viene gestito un page fault?

La gestione del **page fault** avviene nel seguente modo:

- Il **sistema operativo** decide se la pagina che ha generato il fault è non valida o non in memoria:
  - a) **primo caso**: abort del processo;
  - b) **secondo caso**: caricamento pagina da **Backing store**.
- Il **sistema operativo** trova un frame libero e quindi schedula l'operazione di caricamento dalla **Backing store**.
- Al ripristino, aggiorna la **tabella delle pagine** con il frame caricato (e setta il bit di validità).
- Infine ritorna dall'interruzione di **page fault**, e il **processore** riesegue l'istruzione che era stata interrotta.



### 63) Quali sono le politiche di caricamento della pagina?

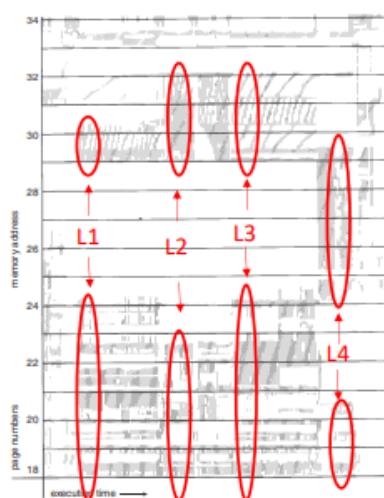
Si hanno due politiche:

- **Paginazione su richiesta pura**: le pagine vengono caricate solo quando servono (ad esempio, all'avvio di un processo non viene caricata alcuna pagina);
- **Prefetching**: vengono caricate anche altre pagine in un "intorno" della pagina richiesta.

La riesecuzione dell'istruzione interrotta da un page fault può essere complessa quando l'istruzione occupa/modifica più parole di memoria su pagine diverse (cosa particolarmente vera per i processori **CISC**)

### 64) Parlare del modello di località?

Secondo il **modello di località** in un certo momento un **processo** opera su una certa località, ossia su un certo sottoinsieme di pagine e i **processi** nel tempo "migrano" da una località all'altra. Le **località** possono sovrapporsi.



### 65) Come vengono gestiti i frame liberi?

I **frame liberi** vengono gestiti attraverso una lista dei **frame liberi** mantenuta dal sistema operativo. Essi vanno azzerati (**zero-fill**) prima di essere assegnati

ad un processo per evitare **leak** di informazioni e man mano che i **processi** richiedono **frame** la lista si riduce: se si azzera, o scende sotto una certa dimensione minima, occorre ripopolarla.

### 66) Come si calcola il tempo medio di accesso effettivo?

Siano:

→ **p** = probabilità di un page fault (% page fault per istruzioni eseguite);

→ **ma** = tempo medio di accesso alla memoria;

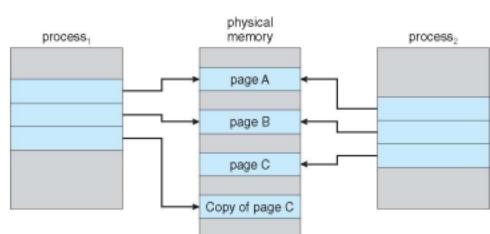
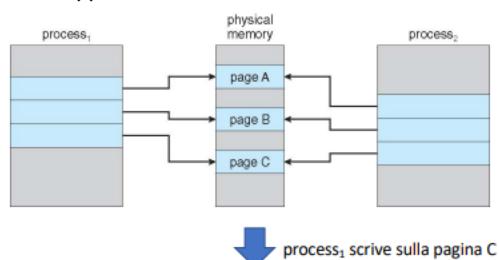
→ **pf** = tempo medio di gestione del page fault.

Il **tempo medio di accesso effettivo** è:

$$\text{EAT} = (1 - p) * \text{ma} + p * \text{pf} = \text{ma} + p * (\text{pf} - \text{ma}) \approx \text{ma} + p * \text{pf}$$

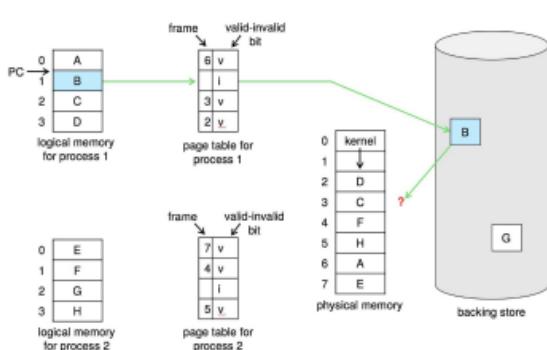
### 67) Cosa si intende per Copy - On - Write?

L'approccio **Copy - On - Write** permette ad un **processo figlio** di condividere tutte (non solo quelle **read - only**) le pagine con il **processo padre**. Se uno dei due processi modifica una pagina condivisa, la pagina viene copiata. Utile per la `fork()`.



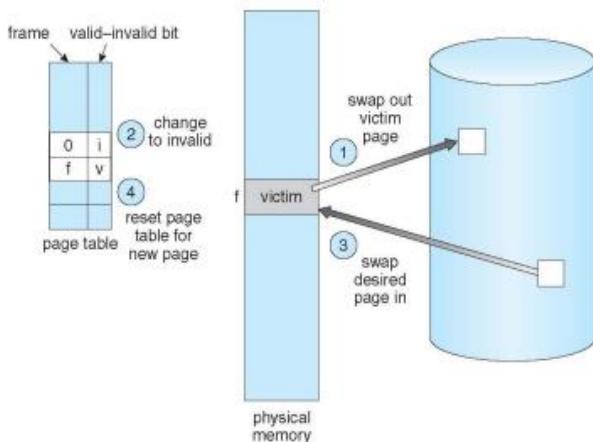
### 68) Cosa consiste l'approccio di sostituzione di pagina?

L'approccio di **sostituzione di pagina** consiste nel liberare un frame attualmente inutilizzato.



La **routine di page fault** aggiornata:

- si individua su disco la locazione della pagina richiesta;
- si cerca un frame libero;
- se esiste viene selezionato, altrimenti si usa l'**algoritmo di sostituzione delle pagine** per selezionare il **frame vittima** effettuando così il **page out** del contenuto del frame vittima e si aggiornano le tabelle delle pagine e dei frame;
- si effettua **page in** della pagina richiesta nel frame libero;
- si aggiornano le **tabelle delle pagine e dei frame**.



### 69) E' possibile ottimizzare l'approccio di sostituzione di pagina?

Si, è possibile ottimizzare l'approccio e l'ottimizzazione consiste in:

- non fare **page out** delle pagine **read - only**;
- le **entry** della tabella delle pagine hanno di solito un **bit di modifica**, che l'hardware imposta a **1** se una certa pagina viene modificata dopo il suo caricamento: le pagine con il **bit di modifica** a **0** non necessitano di **page out**.

### 70) Differenza tra algoritmi di allocazione frame e sostituzione pagine

L'**algoritmo di allocazione frame** determina quanti **frame** assegnare ad ogni processo, mentre l'**algoritmo di sostituzione pagine** determina qual è il **frame vittima** quando è necessaria una **sostituzione di pagina**.

### 71) Differenza tra allocazione fissa e allocazione variabile

Nell'**allocazione fissa** il numero di frame allocati a ciascun processo non cambia nel corso dell'esecuzione e si ha semplicità, predicitività della quantità di **memoria** dedicata a ciascun **processo** (**vantaggi**) ma le **esigenze di memoria** di un processo possono cambiare nel corso dell'esecuzione, e quindi può portare ad un utilizzo inefficiente della memoria (**svantaggio**). Nell'**allocazione variabile** il numero di frame allocati a ciascun processo può cambiare nel corso dell'esecuzione e si ha complessità, non predicitività della quantità di memoria.

dedicata a ciascun processo (**svantaggio**) ma migliore utilizzo della memoria (**vantaggio**).

### 72) Differenza tra sostituzione globale e locale

Nella **sostituzione globale** il frame vittima è selezionato tra tutti i frame di tutti i processi e un processo può perdere pagine a causa degli altri processi  $\Rightarrow$  il tempo medio di accesso alla memoria per un certo processo dipende dal carico (**svantaggio**) ma si ha un utilizzo efficiente della memoria (**vantaggio**). Nella **sostituzione locale** il **frame vittima** è selezionato tra i frame del processo che ha generato il **page fault** ed ogni processo ha sempre lo stesso numero di pagine  $\Rightarrow$  il tempo medio di accesso alla memoria per un certo processo non dipende dal carico (**vantaggio**) ma si potrebbe avere un sottoutilizzo della memoria (**svantaggio**).

### 73) Quando un processo è in thrashing?

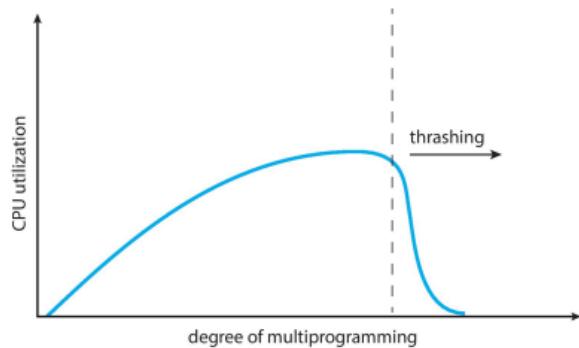
Un **processo** viene definito in **thrashing** se trascorre più tempo nella gestione dei suoi page fault che non nell'esecuzione del suo programma.

### 74) Perché avviene il thrashing?

Perché ad un certo punto risultano presenti  $n$  processi in memoria con

$$\sum_{i=1}^n \text{locality}_i > \text{memory}$$

Quando questo si verifica, ci sono dei processi in memoria che non hanno la propria località completamente in memoria.



### 75) Come rimediare al thrashing e quali sono le due tecniche utilizzate per individuare tale fenomeno?

Si può rimediare al thrashing individuando quando tale situazione si verifica, e quindi riducendo il **grado di multiprogrammazione**. I due approcci utilizzati per individuare il **thrashing** sono:

- **modello working set**;
- **frequenza dei page fault (locale)**.

### 76) Caratteristiche della page - fault?

Viene calcolata per ciascun processo la frequenza dei suoi page fault (**page fault frequency, PFF**) e si stabilisce il range di accettabilità per la PFF:

- se  $PFF < \text{soglia minima}$ , il processo perde frame;
- se  $PFF > \text{soglia massima}$ , il processo acquista frame.

Se non ci sono abbastanza **frame liberi** per abbassare la **PFF** di un **processo**, lo **scheduler** a medio termine sospende uno o più **processi vittima**.

