Memoria virtuale e sistema operativo

Index

Gestione della memoria: decisioni da prendere

- Usare o no la memoria virtuale?
- Usare solo la paginazione?
- Usare solo la segmentazione?

- Usare paginazione e segmentazione?
- Che algoritmi usare per gestire i vari aspetti della gestione della memoria?

Elementi centrali per il progetto del SO

Quando si progetta un sistema operativo è necessario decidere le seguenti cose:

- Politica di prelievo (*fetch policy*)
- Politica di posizionamento (*placement policy*)
- Politica di sostituzione (*replacement policy*)
- Altro (gestione del resident set, politica di pulitura, controllo del carico)
  Il tutto, cercando di minimizzare i page fault; non c'è una politica sempre vincente

Fetch policy

Decide quando una pagina data deve esser portata in memoria principale.

Si usano principalmente due politiche:

- paginazione su richiesta (*demand paging*)
- prepaginazione (*prepaging*)

Demand paging

Una pagina viene portata in memoria principale nel momento in cui qualche processo la richiede. Ciò causa molti page fault nei primi momenti di vita del processo

Prepaging

Cerca di anticipare le necessità del processo. Questa politica infatti porta in memoria principale più pagine di quelle richieste (ovviamente si tratta di pagine vicine a quella richiesta)

Placement policy

La placement policy decide in quale frame mettere una pagina una volta che è stata prelevata dal disco. Seppur tramite la traduzione degli indirizzi la si può posizionare ovunque, tipicamente una pagina viene posizionata nel **primo** (con indice numericamente più basso) **frame libero**.

> **Hint**
>
> Questa politica si applica quando ci sta almeno un frame libero in RAM, se non ne è disponibile nessuno si parlerà di *replacement policy*

Replacement policy

Questa viene applicata quando è stato prelevato una pagina dal disco, ma non è disponibile alcun frame in RAM in cui posizionarla
Essenzialmente, una volta deciso quale è il frame giusto da sostituire tramite un algoritmo di replacement policy (generalmente si cerca di minimizzare la possibilità che la pagina appena sostituita venga richiesta di nuovo, usando il principio di località, si cerca di predire il futuro sulla base del passato recente) e inoltre è necessario aggiornare la tabella della pagine. Nella pagina prelevata dal disco viene impostato il bit di presenza a uno mentre per la pagina da sostituire il bit di presenza viene impostato a zero

Frame bloccati

Bisogna tenere presente, nella replacement policy, che alcuni frame potrebbero essere bloccati, attraverso un bit hardware gestito dal SO. Tipicamente questo stato riguarda frame del sistema operativo stesso, oppure di processi che potrebbero riguardare trasferimenti di I/O

Gestione del resident set

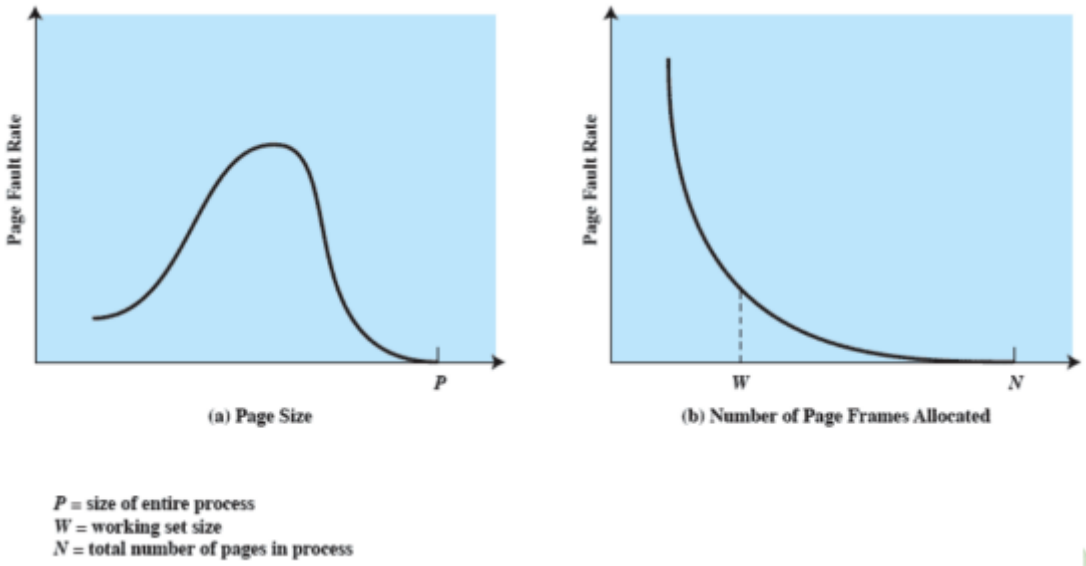
La gestione del resident set risponde a 2 necessità:

- decidere per ogni processo che è in esecuzione quanti frame vanno allocati (*resident set management*)
- decidere se, quando si rimpiazza un frame, è possibile sostituire solo un frame che fa parte dello stesso processo oppure se si può sostituire un frame qualsiasi (*replacement scope*)

Resident set management

Sono presenti due possibilità per decidere quanto spazio dedicare ad ogni singolo processo in RAM:

- **allocazione fissa** → il numero di frame è deciso a tempo di creazione del processo
- **allocazione dinamica** → il numero di frame è deciso durante la vita del processo (magari basandosi su statistiche che man mano vengono raccolte)



Ovviamente, con resident set alto si ha ottimi page fault rate, ma poca multiprogrammazione

Replacement scope

Anche qui si hanno due possibilità:

- **politica locale** → se bisogna rimpiazzare un frame, si sceglie un altro frame dello stesso processo
- **politica globale** → si può scegliere qualsiasi frame (ovviamente non del SO

In tutto si hanno 3 possibili strategie, infatti con l'allocazione fissa, la politica globale non si può usare altrimenti si potrebbe ampliare il numero di frame do un processo, e non sarebbe più allocazione fissa

---

Politica di pulitura

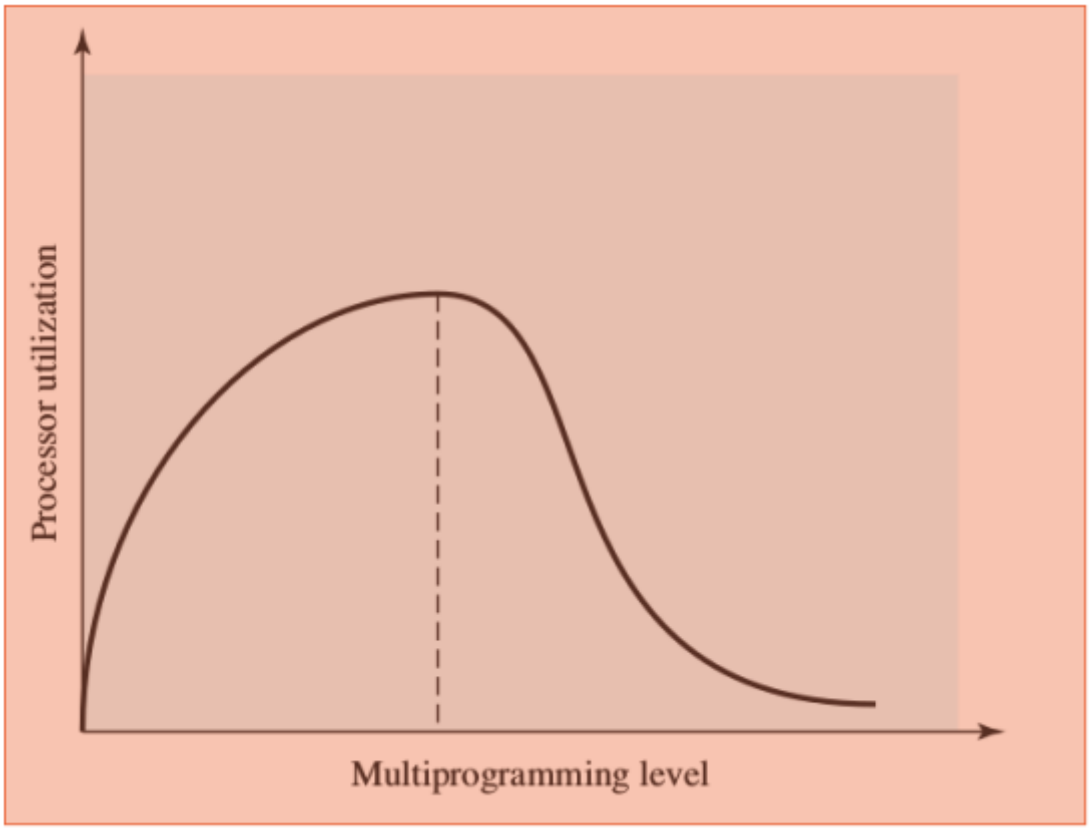
Se un frame è stato modificato, va riportata la modifica anche sulla pagina corrispondente.
Anche qui si hanno due possibilità per decidere quando riportare questa modifica:

- non appena avviene la modifica
- non appena il frame viene sostituito

Tipicamente si fa una via di mezzo, intrecciata con il *page buffering* (concetto di I/O che vedremo); solitamente, si raccolgono un po' di richieste di frame da modificare e le si esegue

---

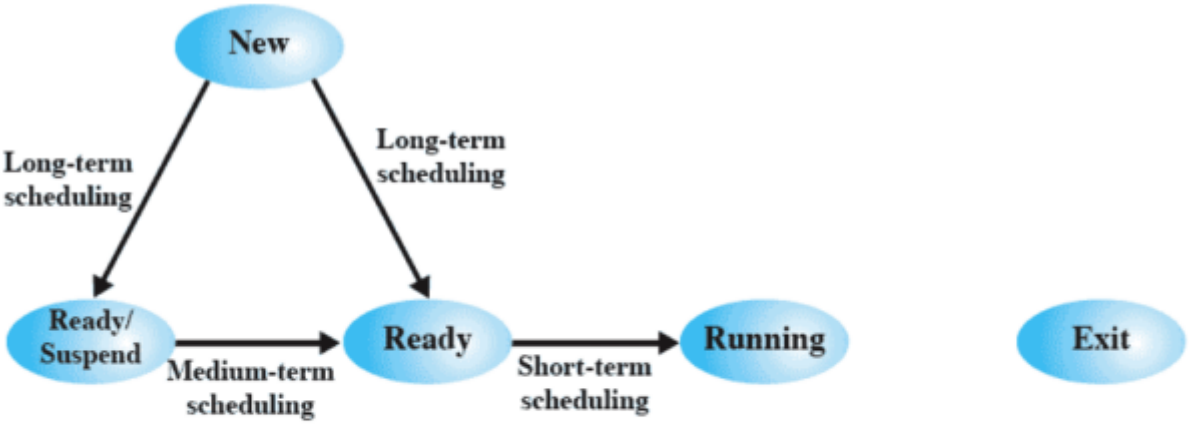
Controllo del carico (medium term scheduler)

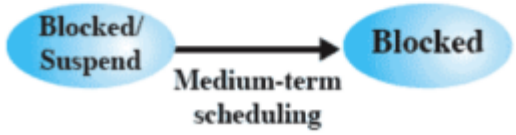


Lo scopo del *medium term scheduler* è quello di **controllo del carico**, ovvero mantenere il livello di multiprogrammazione il più alto possibile ma senza arrivare al trashing (ottimizzando il page fault rate).
Per farlo vengono usate delle **politiche di monitoraggio** che si occupano di prendere la decisione; una tecnica tipica è quella di misurare il tempo tra 2 fault di pagina e confrontarlo con il tempo medio di gestione di un fault (se sono troppo vicini i due valori siamo prossimi al trashing, se sono troppo distanti vuol dire che sto usando poco il processore e che posso aumentare il livello di multiprogrammazione). Questa viene invocata ogni tot page fault e fa parte dell'algoritmo di rimpiazzamento

Per farlo il medium term scheduler ha due possibilità: o sospendere un processo, oppure metterlo in RAM.

Stati dei processi e scheduling





Adesso possiamo specificare meglio cosa vuol dire che un processo è suspended, vuol dire che il suo resident set è zero (non ci sono pagine in RAM). Mentre ready vuol dire che una parte del processo è in RAM (almeno una pagina).
Uno dei motivi per cui un processo diventa suspended è a causa del medium term scheduler.

Come scegliere un processo da sospendere?

Abbiamo varie possibilità per scegliere il processo da sospendere nel caso in cui si è prossimi al trashing e quindi è necessario abbassare il livello di multiprogrammazione:

- processo con minore priorità
- processo che ha causato l'ultimo page fault
- ultimo processo attivato
- processo con il working set più piccolo
- processo con immagine più grande (più grande numero di pagine)
- processo con il più alto tempo rimanente di esecuzione (se disponibile)

Algoritmi di sostituzione

Per la replacement policy esistono vari algoritmi di sostituzione:

- sostituzione ottima
- sostituzione della pagina usata meno di recente (*LRU*)

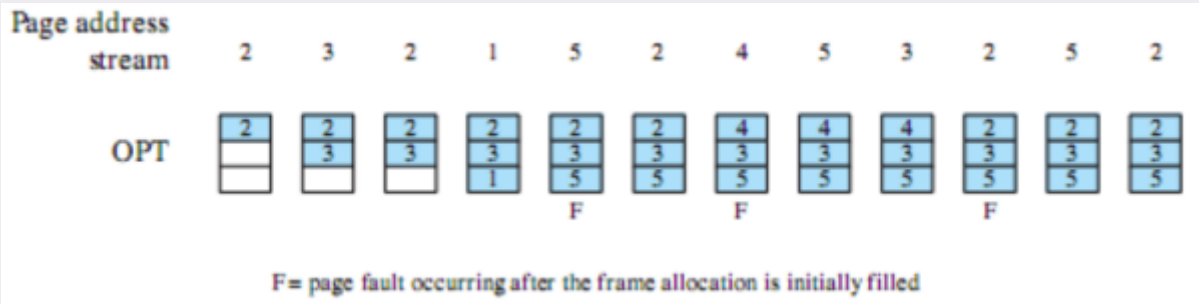
- sostituzione a coda (FIFO)
- sostituzione ad orologio (clock)

> **Note**
>
> Gli esempi riportati nel seguito usano tutti la stessa sequenza di richiesta a pagine:
>
> $$2\ 3\ 2\ 1\ 5\ 2\ 4\ 5\ 3\ 2\ 5\ 2$$
>
> Si suppone inoltre che ci siano solo 3 frame in memoria principale

Sostituzione ottimale

Con la sostituzione ottimale si sostituisce la pagina che verrà richiesta più in là nel futuro. Non è una soluzione implementabile ma è definibile sperimentalmente

> **Example**



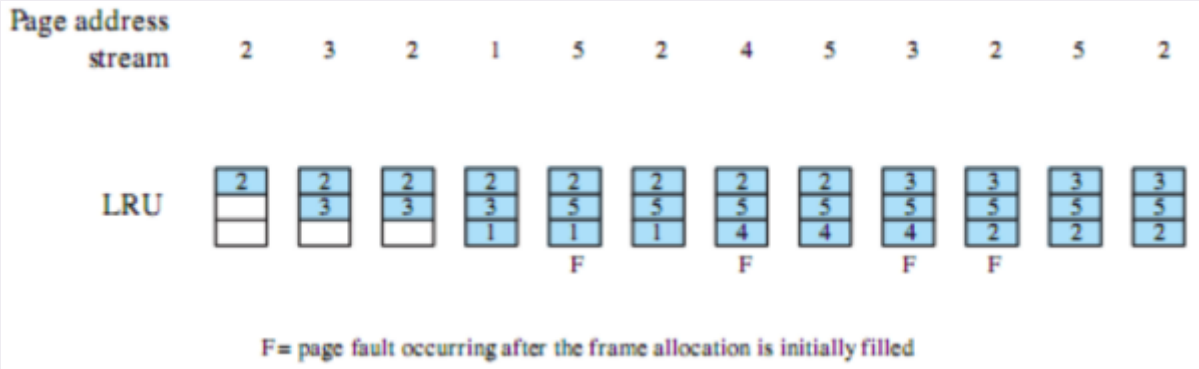
Risultato: 3 page faults

Sostituzione LRU

L'algoritmo LRU sostituisce la pagine cui non sia stato fatto riferimento per il tempo più lungo.
Basandosi sul principio di località, dovrebbe essere la pagina che ha meno probabilità di essere usata nel prossimo futuro

La sua implementazione però risulta problematica in quanto occorre etichettare ogni frame con il tempo dell'ultimo accesso (la cache utilizza questa tecnica perché implementata a livello hardware, ma un'implementazione del genere per la RAM risulterebbe troppo costoso)

> **Example**



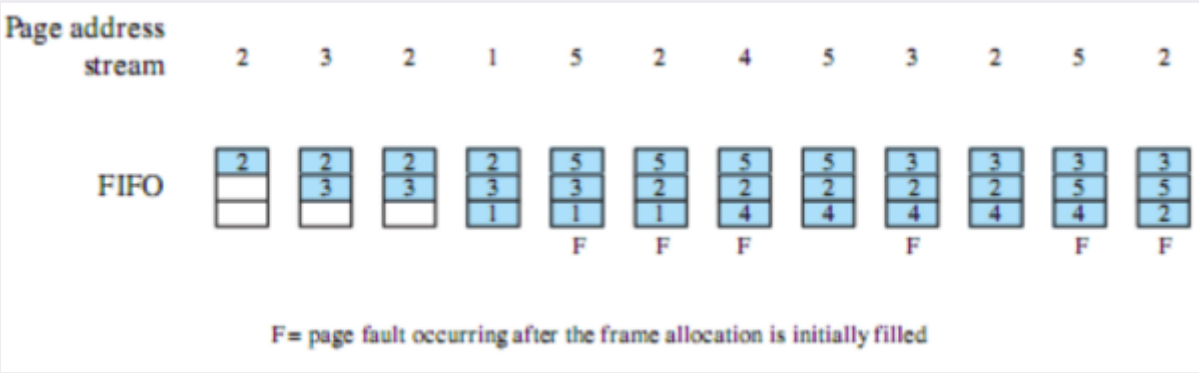
Risultato: 4 page faults

Sostituzione FIFO

L'algoritmo FIFO tratta i frame allocati ad un qualche processo come una coda circolare. Da questa coda, le pagine vengono rimosse a turno (*round robin*)

La sua implementazione risulta semplice, vengono infatti rimpiazzate le pagine che sono state in memoria per più tempo (non necessariamente sono quelle con il minor numero di accessi)

Example

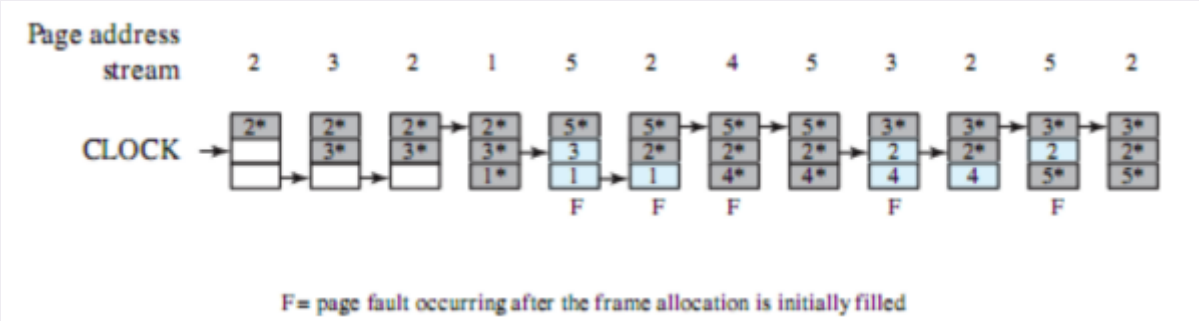


Risultato: 6 page faults

Sostituzione dell'orologio

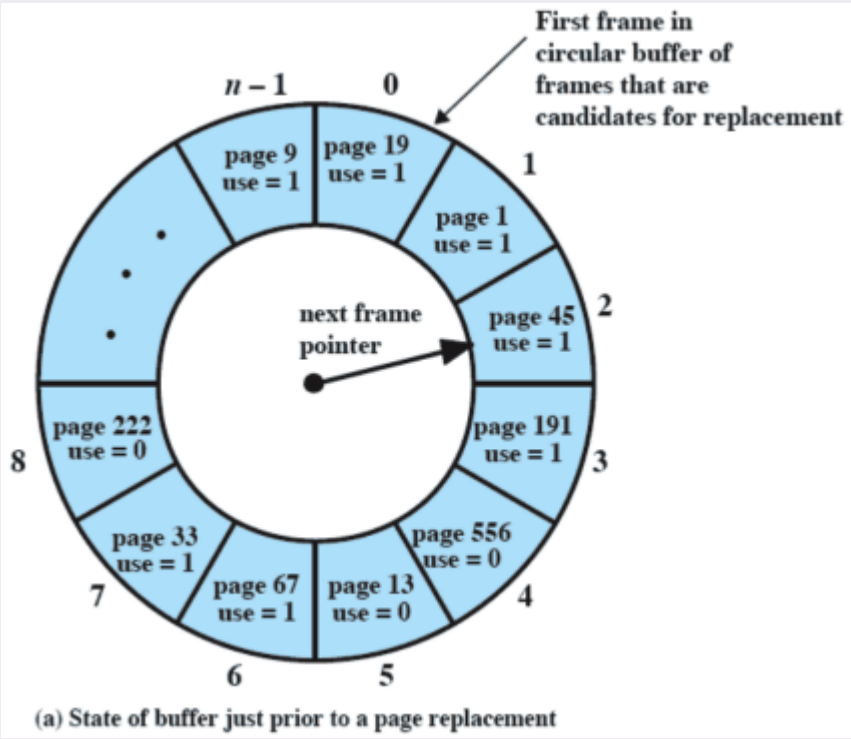
L'algoritmo ad orologio risulta essere il compromesso tra LRU e FIFO.
C'è dunque uno *use bit* per ogni frame, che indica se la pagina caricata nel frame è stata riferita. Il bit è settato ad 1 quando la pagina viene caricata in memoria principale, e poi rimesso ad 1 per ogni accesso al suo interno.
Quando occorre sostituire una pagina, il SO cerca come nella FIFO, ma seleziona il frame contenente la pagina che ha per prima lo *use bit* a 0. Se invece si incontra una pagina che lo ha ad 1, lo mette a zero e procede con la prossima

Example

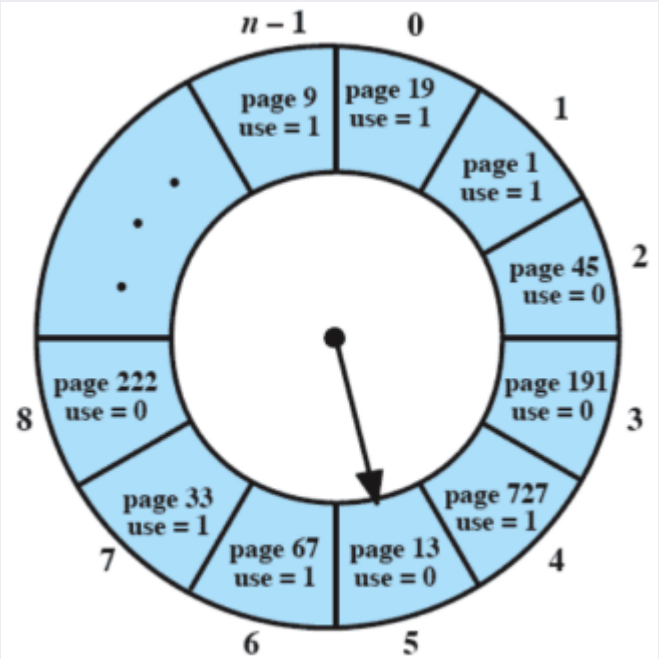


Risultato: 5 page faults

Example

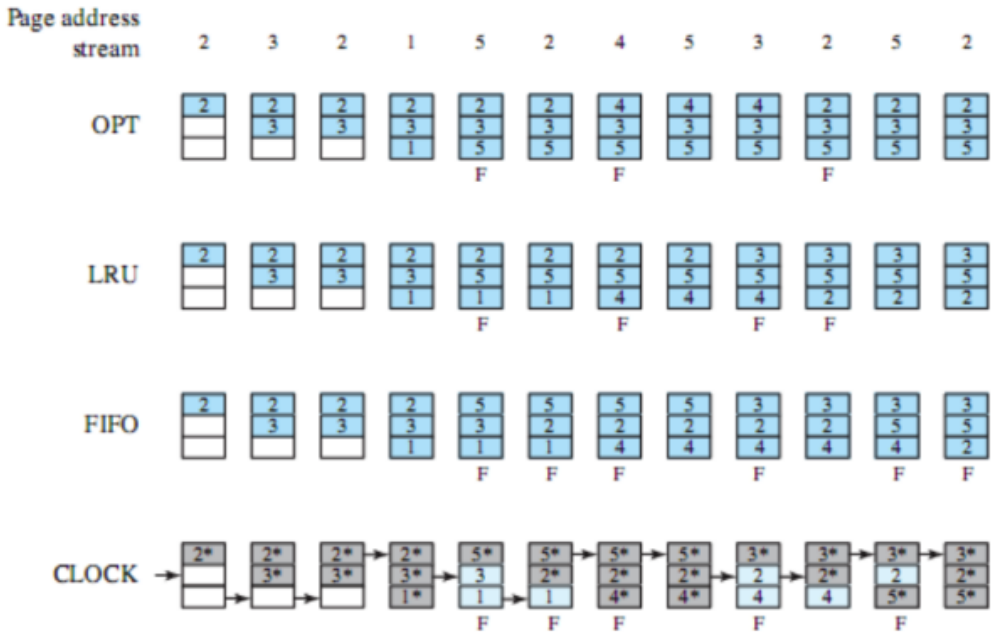


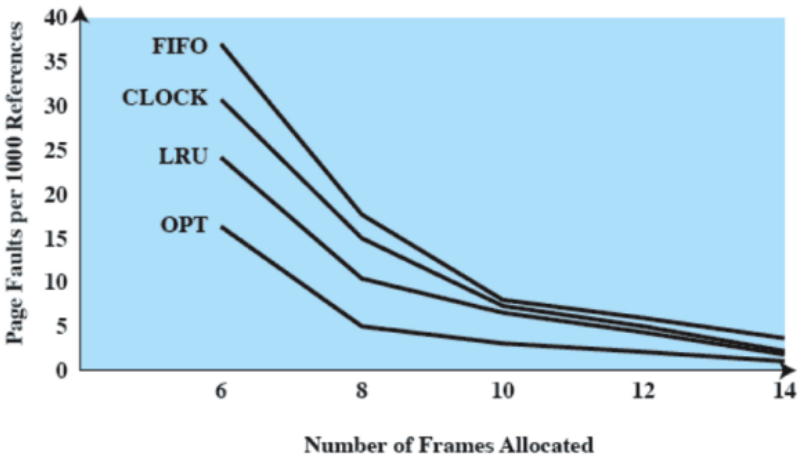
(a) State of buffer just prior to a page replacement



(b) State of buffer just after the next page replacement

Algoritmi a confronto





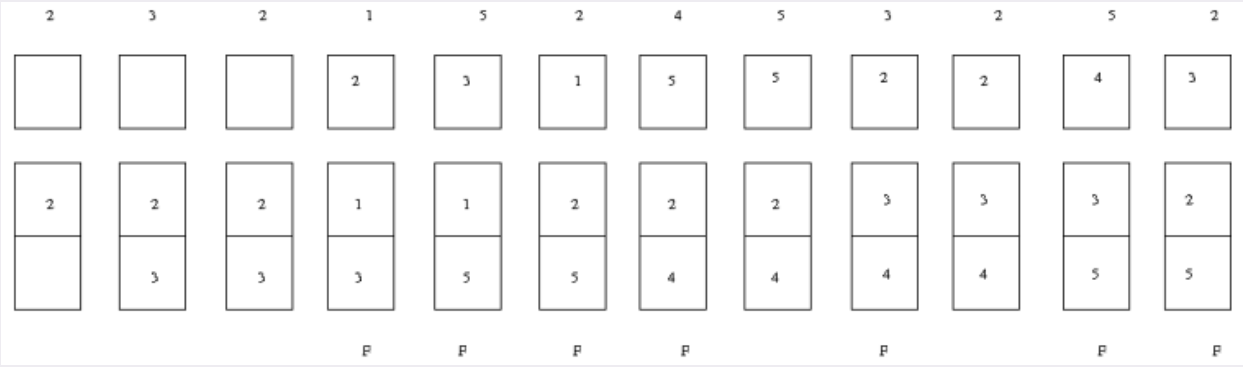
Buffering delle pagine

Questa tecnica è un'ennesima cache (software) per le pagine.
E' nata come una modifica dell'algoritmo FIFO però talvolta è utilizzata anche con LRU e/o clock. Come scopo ha quello di avvicinare il FIFO semplice al clock semplice come prestazioni.
Viene riservata un po' di memoria in meno al processo per poterne conservare una parte per la "cache". Dunque se occorre rimpiazzare una pagina, non viene subito buttata via, ma viene messa in questa cache; così se poi viene nuovamente referenziata, si può subito riportarla in memoria.

Tipicamente la cache è divisa tra pagine modificate e non. Si cerca infatti di scrivere (su disco) le pagine modificate tutti insieme nel momento in cui la dista delle pagine diventa piena o quasi

Example

Assumendo che tutti gli accessi siano in lettura, e che la cache sia di una sola pagina



Risultato: 7 page fault