# Sistemi Operativi
# AA 2019/20
# Esercitazione

06 Maggio 2020

## Esercizio 1

Come si puo' implementare l'algoritmo di sostituzione delle pagine *Second Chance*?

**Soluzione** L'algoritmo *Second Chance* - o altresi' noto come *clock algorithm* - e' un algoritmo FIFO per la sostituzione delle pagine che usa un reference bit (implementato in hardware) per capire quale pagine bisogna rimpiazzare. In particolare:

- se il reference bit di una pagina e' 0 allora essa viene rimpiazzata;
- se il reference bit e' 1 il bit viene settato a 0 e la pagina viene lasciata in memoria, quindi si passa alla pagina successiva - usando le stesse regole.

Un modo per implementare tale algoritmo e' attraverso una *coda circolare* con un puntatore alla pagina da rimpiazzare - che scorre nella coda finche' non trova una pagina con reference bit pari a 0. Trovata una pagina che soddisfa le richieste, viene eliminata dalla coda e rimpiazzata con la nuova. Il funzionamento dell'algoritmo e' illustrato in Figura 1.

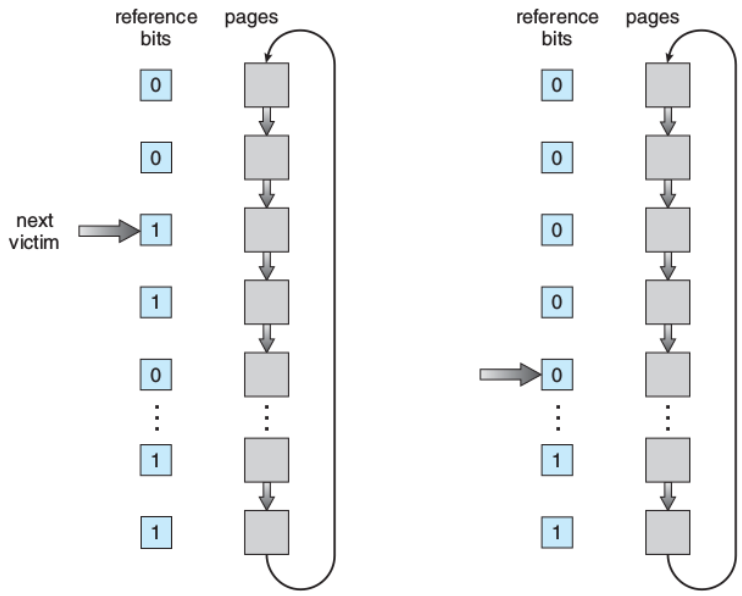



Figure 1: L'algoritmo di sostituzione delle pagine *Second Chance*. Il puntatore scorre finche' una pagina con reference bit 0 non viene trovata; quindi la nuova pagina prendera' il posto di quest'ultima nella coda circolare.

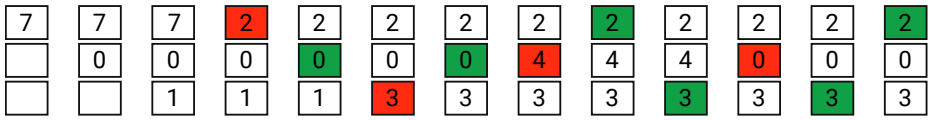
# Esercizio 2

Sia data la seguente traccia di accesso alle pagine di memoria:
7 - 0 - 1 - 2 - 0 - 3 - 0 - 4 - 2 - 3 - 0 - 3 - 2.

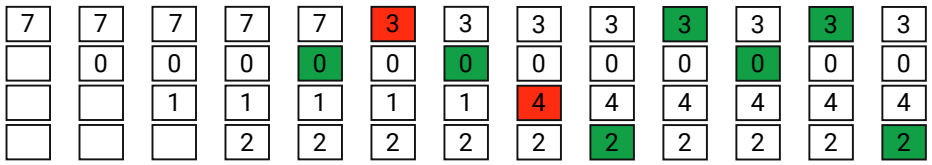
Si assima di avere un Translation Lookaside Buffer (TLB) di 3 elementi, gestito con politica Optimal Replacement (OR). Si assuma che un ciclo di fetch duri $T_{\text{fetch}}$ ed un accesso al TLB impieghi $T_{\text{TLB}}$.

**Domanda** Di quanto aumentano le prestazioni con un TLB a 4 elementi?

**Soluzione** Le tracce di accesso sono riportate rispettivamente in Figura 2a e Figura 2b.




(a) Traccia con TLB di 3 elementi.




(b) Traccia con TLB di 4 elementi.

Figure 2: Tracce d'accesso con TLB a 3 e 4 elementi. In rosso sono evidenziate le sostituzioni, in verde gli *hit*.

Nel caso di TLB a 3 elementi il numero di hit e' pari a $N_{hit} = 6$, con una percentuale di *page fault* del 54%; passando a 4 elementi si avra' $N_{hit} = 7$ ed una percentuale di *page fault* del 46%. Il guadagno prestazionale e' quindi pari a circa l'8%.

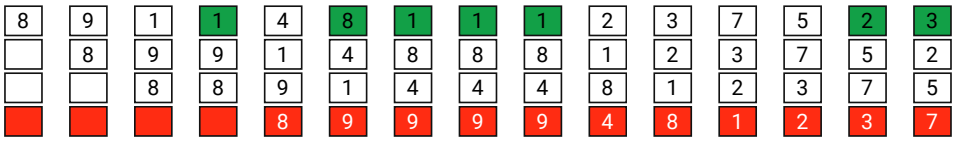
# Esercizio 3

Sia data la seguente traccia di accesso alle pagine di memoria:

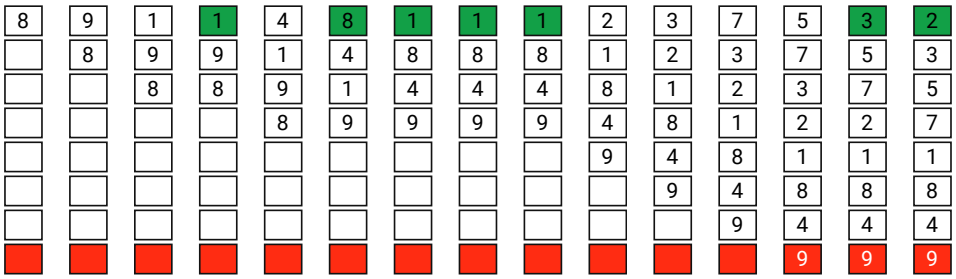
8 9 1 1 4 8 1 1 1 2 3 7 5 2 3.

**Domanda** Si assuma di avere un TLB di 4 slot gestito in modalita' Last Recently Used (LRU). Di quanto migliorano le prestazioni della memoria raddoppiandolo?

**Soluzione** La tracce di accesso nei due casi sono raffigurate rispettivamente nelle Figure 3a e 3b.




(a) Traccia con TLB a 4 slot




(b) Traccia con TLB a 8 slot

Figure 3: Illustrazione della traccia di accesso alle pagine di memoria. In verde indichiamo un *hit* nel TLB, mentre in rosso viene indicata la pagina candidata ad eliminazione - secondo la politica LRU.

Quindi, supponendo che il TLB abbia 4 unita', il tempo di accesso medio $T^4_{mean}$ e' dato dalla seguente relazione:

$$T^4_{mean} = 7T_{hit} + 8Tmiss$$

dove

$$T_{hit} = T_{TLB} + T_{fetch}$$
$$T_{miss} = 2T_{TLB} + 2T_{fetch}$$

Analogamente, nel caso in cui il TLB abbia 8 unita', il tempo medio sara' calcolato come segue:

$$T^8_{mean} = 7T_{hit} + 8Tmiss$$

E' facilmente intuibile da un semplice confronto che in questo particolare caso aumentare la dimensione del TLB non influenza le prestazioni.

# Esercizio 4

Sia dato un sistema con paginazione con frame di dimensione 200 e TLB a 3 entries, gestito con politica di rimozione della pagine LRU. Si ipotizzi che un piccolo processo occupi completamente la prima e meta' della seconda pagina del sistema - i.e. da locazione 0 a 299. Si consideri quindi un array bidimensionale `int A[][]` con dimensioni $R \times C$ con $R = C = 100$. Si assuma che la matrice sia allocata in memoria in modalita' *column-major*, ovvero:

```
1
2     int** A = (int**)malloc(sizeof(int*)*R);
3     for (int r = 0; r < R; ++r) {
4       A[r] = (int*)malloc(sizeof(int)*C);
5     }
6
```

Calcolare il numero di *page fault* dovute all'inizializzazione dell'array nelle seguenti modalita':

**Modo 1**

```
1
2       for (int r = 0; r < R; ++r)
3         for (int c = 0; c < C; ++c)
4           A[r][c] = 0;
5
```

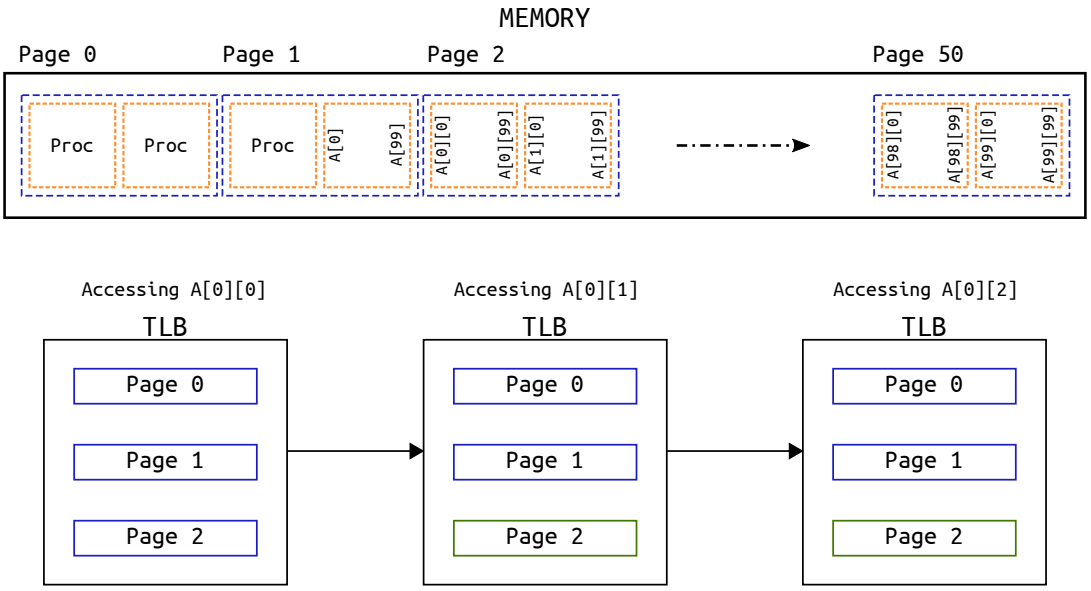
**Modo 2**

```
1
2       for (int c = 0; c < C; ++c)
3         for (int r = 0; r < R; ++r)
4           A[r][c] = 0;
5
```

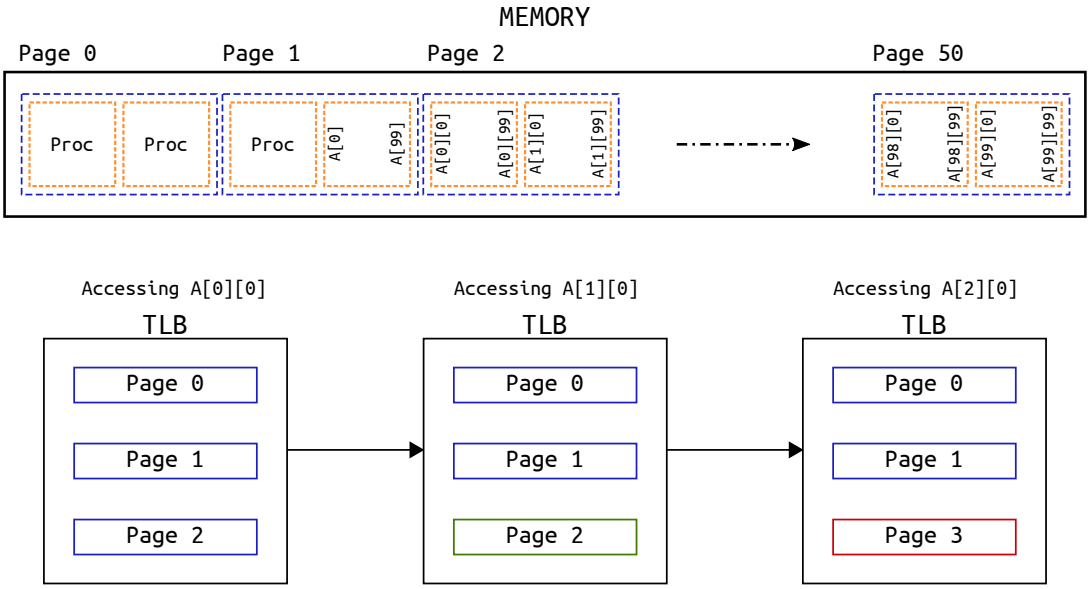
**Soluzione** Le traccia di accesso alle pagine nel primo e nel secondo caso sono illustrate rispettivamente nella Figura 4a e Figura 4b.

Come illustrato in Figura 4, nella modalita' 1 accederemo alle pagine in maniera sequenziale - per come e' allocata la memoria. Nella modalita' 2, l'accesso alla memoria sara' scattered, forzando a swappare pagina ogni 2 accessi. Da qui avremo che il numero di *page fault* sara'

- ⋆ Modalita' 1 $\rightarrow N_{fault} = 50$
- ⋆ Modalita' 2 $\rightarrow N_{fault} = 5000$




(a) Snapshot del TLB durante inizializzazione per colonne




(b) Snapshot del TLB durante inizializzazione per righe

Figure 4: Illustrazione della traccia di accesso alle pagine in memoria. Si noti che inizializzando per *righe* - i.e. modalita' 2 - richiede uno swap di pagina ogni due accessi.

## Esercizio 5

Siano dati due processi `P1` e `P2`. Si supponga che la loro traccia di accesso alle pagine in memoria sia quella illustrata nella Figura 5. Si supponga che ogni pagina abbia dimensione `4MB`, che la memoria fisica disponibile sia `8MB` e che sia presente un disco fisico meccanico da 16 GB dedicato allo swap. Cosa succede nell'istante di tempo $t = 5$?

**Soluzione** In questo caso, il nostro sistema potra' contenere al massimo 2 pagine di memoria in RAM, ergo $D_{RAM} = 2$, mentre i *working set* dei due processi all'istante $t = 5$ avranno dimensione $\texttt{WSS}_1 = 2$ e $\texttt{WSS}_2 = 2$. Cio' implica che si verifichera' il fenomeno noto come *thrashing*, ovvero la CPU sara' impiegata piu' tempo nello *swapping-in/out* delle pagine che nell'esecuzione del processo stesso. Cio'

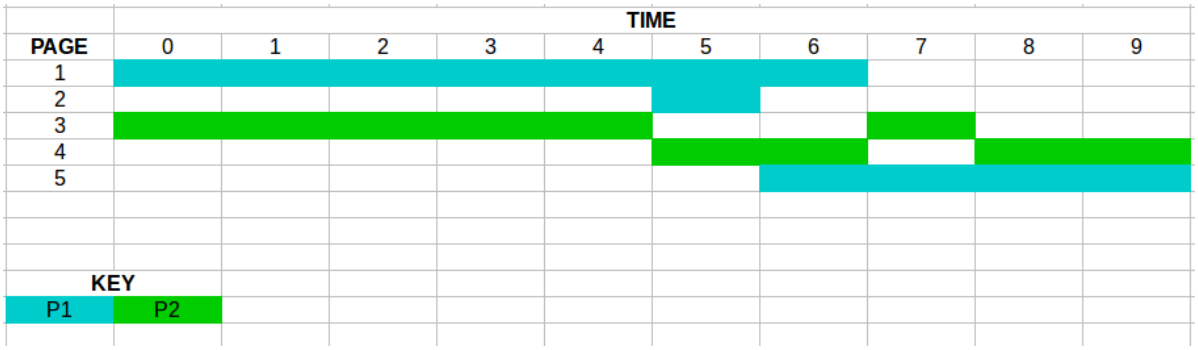



Figure 5: Traccia di accesso alle pagine dei processi `P1` e `P2` - rispettivamente in ciano e verde.

poiche' il working set dei due processi non puo' essere contenuto in RAM, in formulæ:

$$\sum_{p} \texttt{WSS}_p > D_{\texttt{RAM}}$$

Per evitare tale fenomeno, bisognerebbe aumentare la dimensione della RAM almeno fino a `16MB` - lo spazio minimo necessario a contenere 4 pagine da `4MB` - oppure killare/sospendere un processo.