Esercitazione 4 – Architettura dei Sistemi di Elaborazione – 16/01/2017
Soluzioni

Gestione della memoria, paginazione e gestione del disco

1. **Nell'ambito della gestione della memoria con liste concatenate, considerando una lista parzialmente piena di questo tipo (per ogni tripla, il primo elemento è un flag per capire se si tratta di un buco o un processo, il secondo è l'indirizzo di partenza e il terzo è la lunghezza dell'elemento):**

   P,0,6,→H,6,3,→P,9,8→P,17,4,→H,21,2,→P,23,6,→H,29,4,X

   Dove viene posizionato il nuovo processo P che occupa 4 blocchi? Dove verrebbe posizionato se invece ne occupasse solo 2, secondo gli algoritmi first fit e best fit?

   La risposta alla prima domanda è estremamente semplice: assumendo (come è facile immaginare) che la lettera P rappresenti un processo e la lettera H rappresenti un buco, c'è solo un buco adatto a contenere il nuovo processo da 4 blocchi. Questo buco è quello partente dall'indirizzo 29. Pertanto il nuovo processo verrà posto all'indirizzo 29.
   Passando alla seconda domanda, qui le cose si complicano: è necessario conoscere come lavorano gli algoritmi **first fit** e **best fit**. Sul libro, "I moderni sistemi operativi", *Tanenbaum*, li trovate a pagina 183 e seguenti (nell'edizione in mio possesso), al paragrafo 4.2.2. Gestione della memoria con liste concatenate.
   Vi basta sapere, ad ogni modo, che:
   - l'algoritmo **first fit**, come il nome potrebbe suggerire, inserisce il nuovo processo nella prima posizione possibile dall'inizio della lista,
   - l'algoritmo **best fit**, invece, inserisce il nuovo processo nella posizione tale da contenerlo nel modo migliore e, dunque, nel buco più piccolo disponibile per contenere il processo.

   Considerato quanto detto finora, è facile vedere che, nel caso del **first fit**, il processo andrebbe a finire nel buco che inizia all'indirizzo 6, generando la seguente lista:

   P,0,6,→P,6,2,→H,8,1,→ P,9,8→P,17,4,→H,21,2,→P,23,6,→H,29,4,X

   Nel caso del **best fit**, invece, il processo andrebbe inserito nel buco che inizia all'indirizzo 21, che ha esattamente dimensione 2. Si noti che, invece, tutti gli altri buchi avrebbero dimesione maggiore. La lista risultante, dunque, sarebbe la seguente:

   P,0,6,→H,6,3,→P,9,8→P,17,4,→P,21,2,→P,23,6,→H,29,4,X

2. **Nell'ambito del rimpiazzamento delle pagine, supponendo di usare l'algoritmo not recently used (NRU), e seguendo la seguente tabella delle pagine,**

Pagina	R	M
A	1	0
B	1	1
C	1	0
D	0	1

E	0	0
F	1	1
G	1	0

nel caso in cui avvenga un *page fault*, quale sarà la pagina che verrà posta su disco? Perché? Se invece considerassimo la tabella come una lista ordinata (nell'ordine A, …,G), cosa succederebbe per l'algoritmo second chance? Quale pagina sarebbe rimpiazzata?

Per rispondere a questa domanda, bisogna, come minimo, sapere cosa faccia l'algoritmo NRU e cosa rappresentino i bit R e M. In generale, sarebbe meglio conoscere tutti gli algoritmi di rimpiazzamento delle pagine, che sul libro (sempre "I moderni sistemi operativi") trovate a pag 197 e successive, al capitolo 4.4. Algoritmi di rimpiazzamento delle pagine. In particolare, qui stiamo trattando quelli relativi ai paragrafi 4.4.2. e 4.4.5. ma potrebbero capitare anche gli altri all'esame. Tornando al nostro esercizio, definiamo, per cominciare i bit R e M.

- **R** viene messo a 1 ogni volta che la pagina viene riferita (cioè letta o scritta)
- **M** viene messo a 1 quando la pagina viene scritta, ovverosia modificata.

Ora, questi bit possono riessere posti a 0 tramite software.
Secondo l'algoritmo NRU, le pagine vengono ordinate secondo una priorità per cui possono essere fatte uscire dalla memoria. Il criterio con cui questa priorità è assegnata è il seguente:

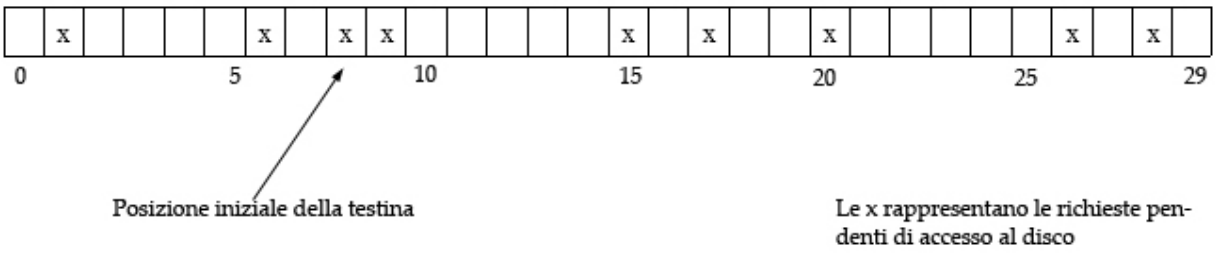
- Priorità per lo swap massima (pagina più conveniente): R=0, M=0, [Classe 0]
- R=0, M=1, [Classe 1]
- R=1, M=0, [Classe 2]
- Priorità per lo swap minima (pagina meno conveniente): R=1, M=1. [Classe 3]

A questo punto, è facile vedere che c'è una sola pagina con massima priorità per lo swap, ed è la pagina denominata E. Dunque, la pagina E verrà posta su disco.
Passiamo, ora, alla seconda domanda. L'algotimo in questione, in questo caso, è l'algoritmo **second chance** o, usando la notazione del libro, **della seconda opportunità**. In questo algoritmo, si trovano nella lista per ordine di ingresso e il bit M non viene considerato. Si scorre, dunque, la lista dall'inizio verso la fine, analizzando il valore del bit R. Se è 1, lo si pone a 0, e si reinserisce la pagina in coda alla lista. Alternativamente, si sposta su disco la pagina.
Considerato quanto detto, applichiamo l'algoritmo. A ha R a 1, quindi va aggiornato a 0 e la pagina va messa in fondo alla coda; lo stesso discorso è applicabile per le pagine B e C. D, invece, ha R a 0 e, dunque, viene spostata su disco. La lista finale, dopo le operazioni descritte, è mostrata in figura.

Pagina	**R**
E	0
F	1
G	1
A	0
B	0
C	0

3. **Nell'ambito della schedulazione delle richieste al disco, si consideri la situazione delle richieste pendenti e della testina illustrata in figura. Come si comporterà la testina nel**

caso in cui l'algoritmo di schedulazione utilizzato sarà l'SSF (Shortest Seek First)? Come si comporterà nel caso in cui sarà l'algoritmo dell'ascensore?



È necessario, in primo luogo, considerare che il tempo necessario per leggere un dato su disco è dato, oltre che dal tempo necessario effettivamente per il trasferimento dei dati, dal tempo necessario per posizionare la testina di lettura sulla locazione corretta del disco. Nel caso in cui le richieste di accesso al disco siano molte, proprio come nell'esercizio, è necessario stabilire l'ordine con cui queste siano risolte, in modo da non avere delay troppo grandi. Per questo, sono stati ideati gli algoritmi di schedulazione del braccio del disco (sul libro, sempre "I moderni sistemi operativi") li trovate a pagina 296 e successivi, al paragrafo 5.4.3. Gli algoritmi di schedulazione del braccio del disco. Nell'esercizio ne usiamo solo due, ma tutti quelli descritti nel paragrafo potrebbero essere richiesti all'esame.
Descriviamo ora gli algoritmi che andremo a utilizzare, anche se il loro funzionamento è facilmente intuibile dal nome.

- L'algoritmo **SSF** risolverà per prime le richieste più vicine alla posizione attuale della testina. Nel caso in figura, la testina si trova alla posizione 8 e ha richieste per le posizioni 1, 6, 8, 9, 15, 17, 20, 26 e 28. La prima risolta, dunque, sarà quella in posizione 8, visto che la testina già si trova nel posto corretto. A seguire, viene risolta quella in posizione 9, visto che, denominati P la posizione della testina e R_k la posizione di tutte le richieste, con k = R_k

  $$\underset{k}{argmin}|(P-R_k)|=9$$

  e $\underset{k}{argmin}|(P-R_k)|$ ci dice proprio la prossima posizione su cui spostarci. Utilizzando la stessa regola, andremo a schedulate le richieste in quest'ordine:
  8, 9, 6, 1, 15, 17, 20, 26 e 28.
- L'algoritmo **dell'ascensore**, invece, funziona proprio come un ascensore in un palazzo con molti piani: si va in una direzione fino alla massima richiesta (in salita, ad esempio), prendendo sull'ascensore tutte le persone in attesa ai piani intermedi, poi si percorre tutto il percorso fino alla minima posizione, risolvendo le richieste via via che vengono sottoposte. Nel caso in cui la spiegazione non sia abbastanza chiara vi rimando comunque al libro.
  Siamo dunque in posizione 8, e assumiamo che ci si stia muovendo in direzione UP (o da sinistra a destra, se preferite). Risolviamo, dunque, tutte le richieste a destra della posizione attuale (risolvendo comunque la 8 per prima). Giunti all'ultima (posizione 28), si ritorna all'indietro, risolvendo sempre tutte le richieste che incontriamo.
  L'ordine delle richieste schedulate, dunque, è: 8, 9, 15, 17, 20, 26, 28, 6, 1.
  Si noti la solidità di questo algoritmo che, anche in caso di aggiunta di richieste (anche molto vicine tra loro), garantisce la *fairness* del sistema, rendendo impossibile la *starvation*.