Tutoriat 8

- Pentru a rula un program, acesta trebuie adus din memorie intr-un proces si rulat
- Memoria principala si registrii sunt singurul tip de storage ce poate fii acessate de CPU
- Intre registrii si Memoria Principala se afla memoria Cache

Paging

- Memoria fizica este impartita in blocuri de marimi fixe , numite **frame-uri** - puteri ale lui 2
- Memoria logica este impartita in blocuri de marimi fixe numite **pagini**
- Se tine cont de toate frame-urile libere
- Pentru a rula un program cu N pagini, avem nevoie de N frame-uri libere si de programul incarcat
- Se creeaza un page table pentru a traduce memoria logica in cea fizica

Swapping

- Un proces poate fii schimbat temporal din memorie intr-o locatie de backup
- Locatia de backup trebuie sa fie destul de mare
- Roll out, roll in - procesele sunt schimbate din memoria fizica in backup in functie de prioritatea lor

Address Translation Scheme

- Adresele generate de CPU sunt impartite in page number si page offset

Page Fault

- Ce este page fault? Daca exista o referinta catre o pagina, dar aceasta nu apare in memorie (sau exista o referinta invalida) - atunci primim page faults

Algoritmii de inlocuire

- Frame allocation algorithm: cate frame-uri sa oferim unui proces
- Ce frame-uri sa inlocuim
- Page replacement: cautam un algoritm ce ne ofera o rata de page-fault minima

Algoritmi:
- LRU: inlocuim paginile ce nu au fost folosite de mult
- OPT: inlocuim paginile ce nu vor fii folosite pentru o perioada cat mai lunga de timp (algoritmul nu este practic ci doar de testare)
- FIFO: Inlocuim paginile una dupa alta in frame-uri

Background

- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are only storage CPU can access directly
- Memory unit only sees a stream of addresses + read requests, or address + data and write requests
- Register access in one CPU clock (or less)
- Main memory can take many cycles, causing a **stall**
- **Cache** sits between main memory and CPU registers
- Protection of memory required to ensure correct operation

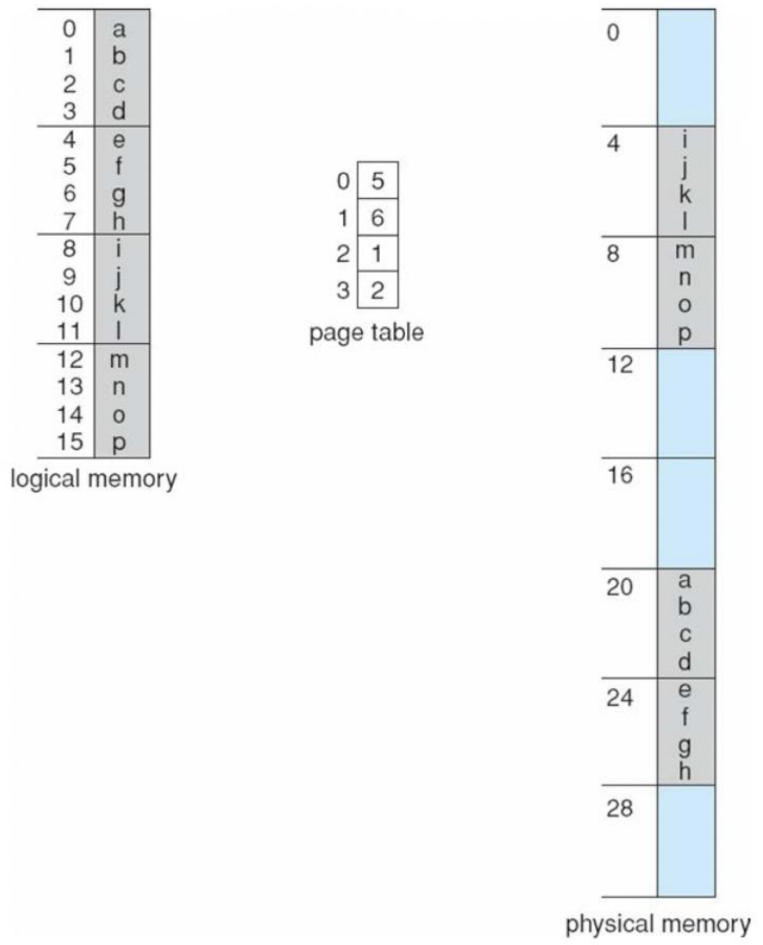
Paging

- Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
  - Avoids external fragmentation
  - Avoids problem of varying sized memory chunks
- Divide physical memory into fixed-sized blocks called **frames**
  - Size is power of 2, between 512 bytes and 16 Mbytes
- Divide logical memory into blocks of same size called **pages**
- Keep track of all free frames
- To run a program of size ***N*** pages, need to find ***N*** free frames and load program
- Set up a **page table** to translate logical to physical addresses
- Backing store likewise split into pages
- Still have Internal fragmentation

Swapping

- A process can be **swapped** temporarily out of memory to a backing store, and then brought back into memory for continued execution
  - Total physical memory space of processes can exceed physical memory
- **Backing store** – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images
- **Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped
- System maintains a **ready queue** of ready-to-run processes which have memory images on disk

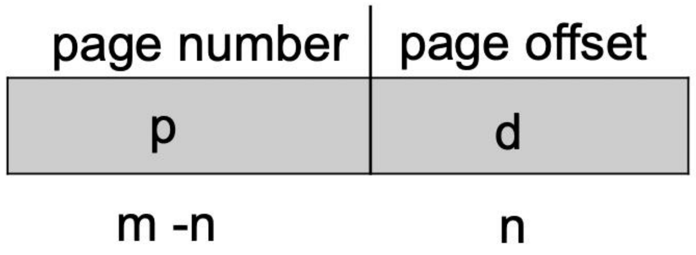
Paging Example



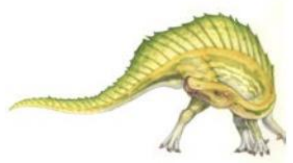
n=2 and *m*=4   32-byte memory and 4-byte pages

Address Translation Scheme

- Address generated by CPU is divided into:
  - **Page number** (*p*) – used as an index into a **page table** which contains base address of each page in physical memory
  - **Page offset** (*d*) – combined with base address to define the physical memory address that is sent to the memory unit



  - For given logical address space 2^m and page size 2^n

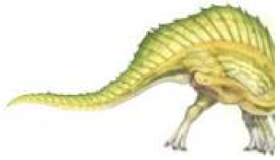


Page Fault

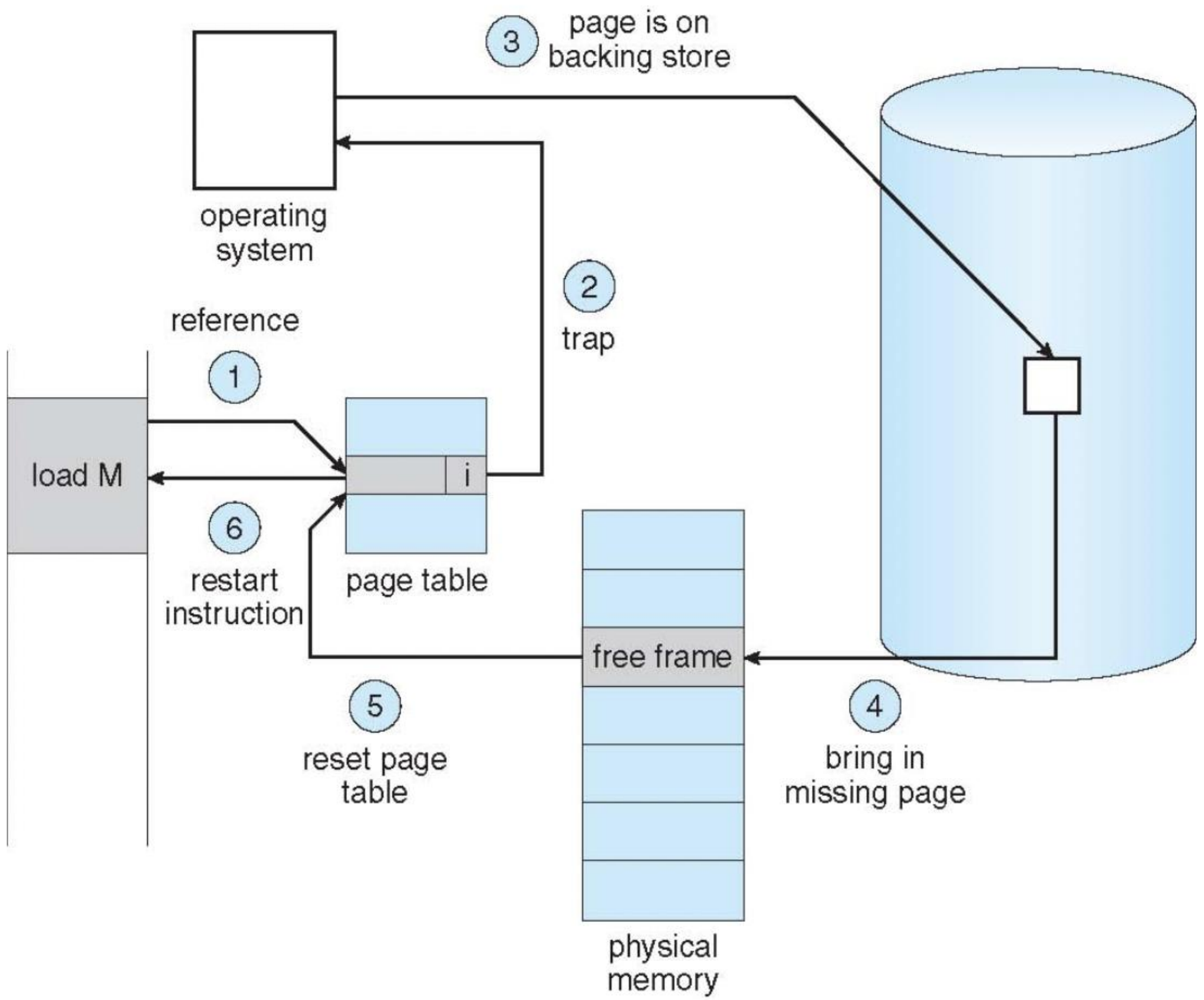
- If there is a reference to a page, first reference to that page will trap to operating system:

  page fault

1. Operating system looks at another table to decide:
   - Invalid reference $\Rightarrow$ abort
   - Just not in memory
2. Find free frame
3. Swap page into frame via scheduled disk operation
4. Reset tables to indicate page now in memory
   Set validation bit = **v**
5. Restart the instruction that caused the page fault



Steps in Handling a Page Fault



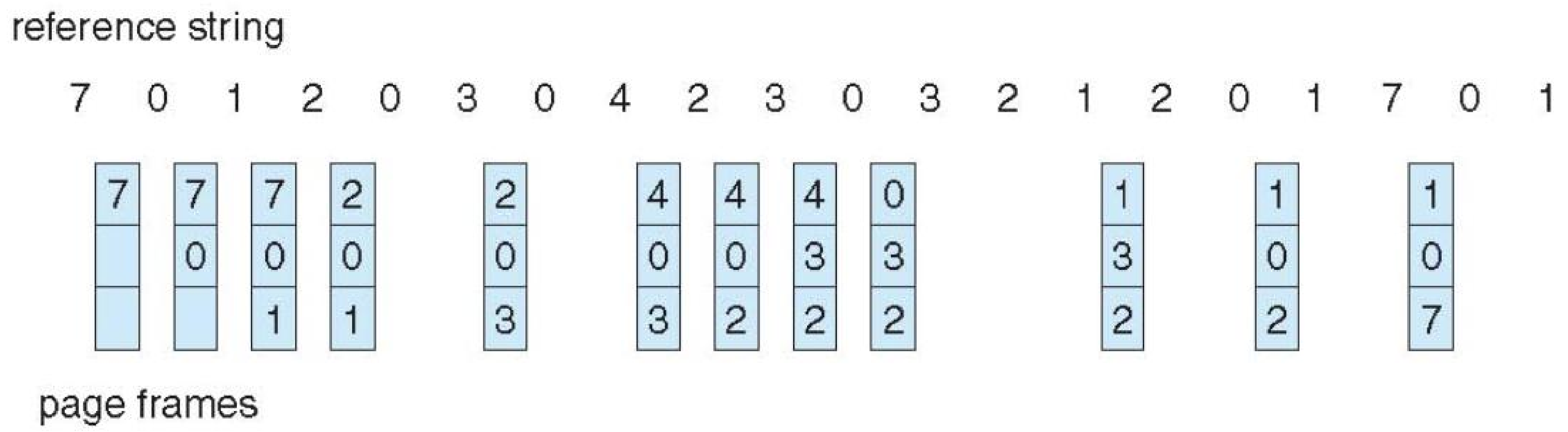
Page and Frame Replacement Algorithms

- **Frame-allocation algorithm** determines
  - How many frames to give each process
  - Which frames to replace
- **Page-replacement algorithm**
  - Want lowest page-fault rate on both first access and re-access
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
  - String is just page numbers, not full addresses
  - Repeated access to the same page does not cause a page fault
  - Results depend on number of frames available
- In all our examples, the **reference string** of referenced page numbers is

  7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1

Least Recently Used (LRU) Algorithm

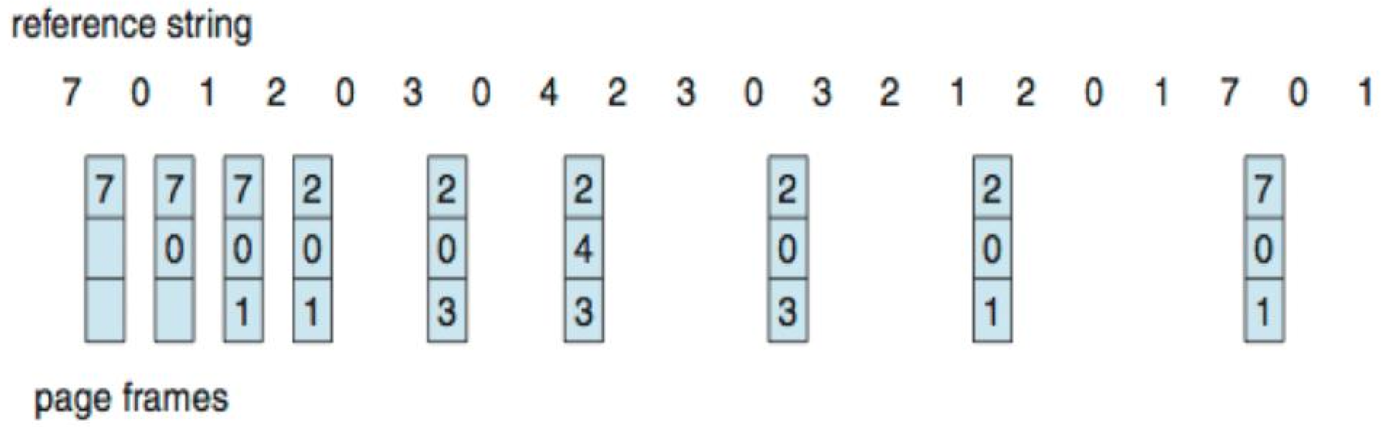
- Use past knowledge rather than future
- Replace page that has not been used in the most amount of time
- Associate time of last use with each page



- 12 faults – better than FIFO but worse than OPT
- Generally good algorithm and frequently used
- But how to implement?

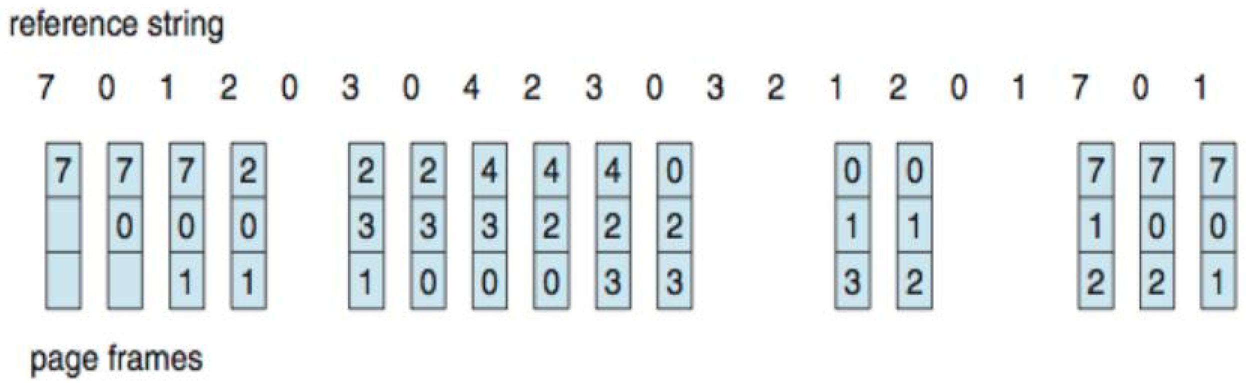
Optimal Algorithm

- Replace page that will not be used for longest period of time
  - 9 is optimal for the example
- How do you know this?
  - Can't read the future
- Used for measuring how well your algorithm performs



First-In-First-Out (FIFO) Algorithm

- Reference string: **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**
- 3 frames (3 pages can be in memory at a time per process)



15 page faults

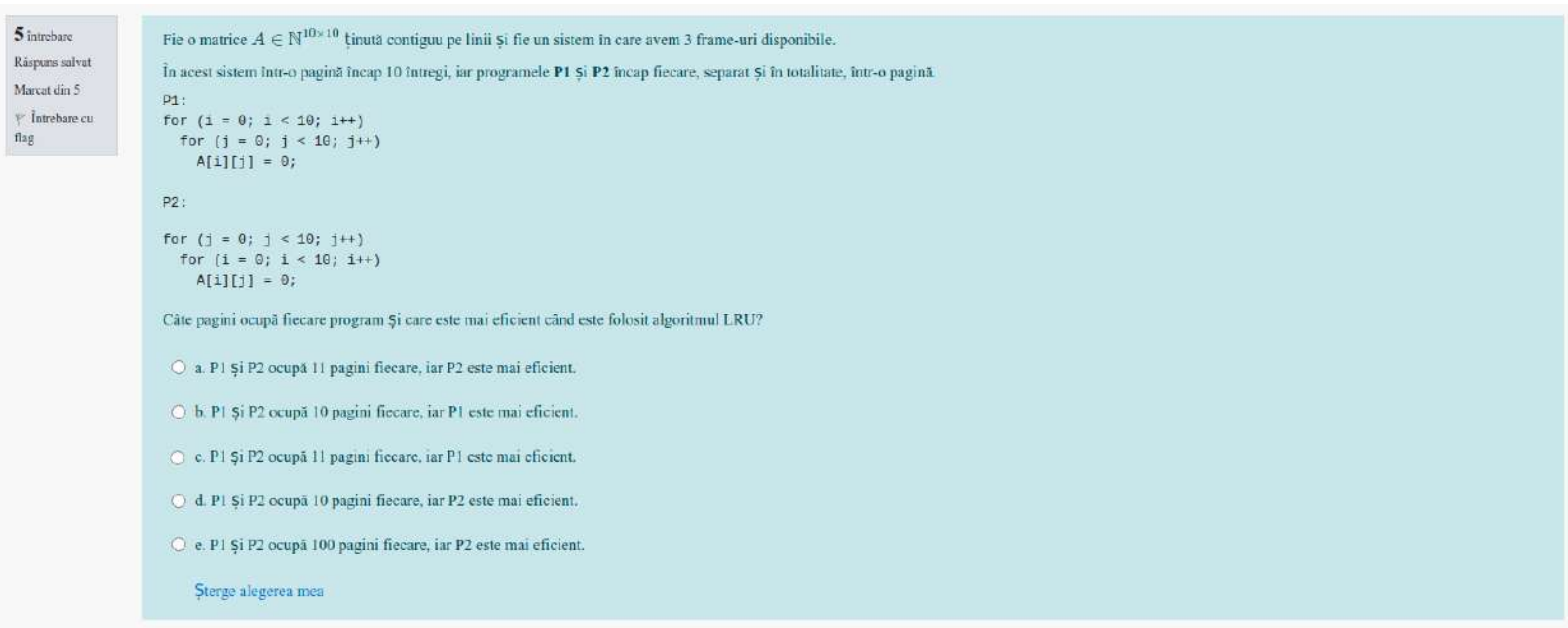
- Can vary by reference string: consider 1,2,3,4,1,2,5,1,2,3,4,5
  - Adding more frames can cause more page faults!
    - **Belady's Anomaly**
- How to track ages of pages?
  - Just use a FIFO queue

9.8 Consider the following page reference string:

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

How many page faults would occur for the following replacement algorithms, assuming one, two, three, four, five, six, or seven frames? Remember all frames are initially empty, so your first unique pages will all cost one fault each.

- LRU replacement
- FIFO replacement
- Optimal replacement



5 Întrebare

Răspuns salvat

Marcat din 5

Întrebare cu flag

Fie o matrice $A \in \mathbb{N}^{10\times 10}$ ținută contiguu pe linii și fie un sistem în care avem 3 frame-uri disponibile.

În acest sistem într-o pagină încap 10 întregi, iar programele **P1** și **P2** încap fiecare, separat și în totalitate, într-o pagină.

```
P1:
for (i = 0; i < 10; i++)
  for (j = 0; j < 10; j++)
    A[i][j] = 0;

P2:

for (j = 0; j < 10; j++)
  for (i = 0; i < 10; i++)
    A[i][j] = 0;
```

Câte pagini ocupă fiecare program și care este mai eficient când este folosit algoritmul LRU?

○ a. P1 și P2 ocupă 11 pagini fiecare, iar P2 este mai eficient.

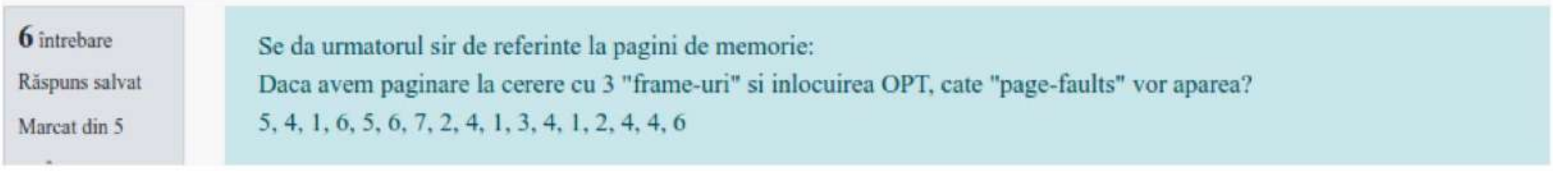
○ b. P1 și P2 ocupă 10 pagini fiecare, iar P1 este mai eficient.

○ c. P1 și P2 ocupă 11 pagini fiecare, iar P1 este mai eficient.

○ d. P1 și P2 ocupă 10 pagini fiecare, iar P2 este mai eficient.

○ e. P1 și P2 ocupă 100 pagini fiecare, iar P2 este mai eficient.

Șterge alegerea mea



6 întrebare

Răspuns salvat

Marcat din 5

Se da urmatorul sir de referinte la pagini de memorie:
Daca avem paginare la cerere cu 3 "frame-uri" si inlocuirea OPT, cate "page-faults" vor aparea?
5, 4, 1, 6, 5, 6, 7, 2, 4, 1, 3, 4, 1, 2, 4, 4, 6