## MEMORIA VIRTUALĂ

Memoria principală poate acţiona ca o memorie cache pentru nivelul de stocare secundar – uzual implementat cu discuri magnetice.

De ce avem nevoie de o memorie virtuală ?

1. Permite folosirea în comun, eficientă şi sigură a memoriei de către mai multe programe

2. Înlăturarea problemelor de programare cauzate de o memorie principală mică

Memoria principală trebuie să conţină doar porţiunile active ale programelor, deci va fi necesar un mecanism de protecţie a programelor între ele – trebuie să ne asigurăm că un program va scrie şi va citi doar din memoria principala atribuită lui.

Memoria virtuală implementează translatarea spaţiului de adrese al programului în adrese fizice. Acestă translatare asigură unicitatea spaţiului de adrese al unui program faţă de alte programe.

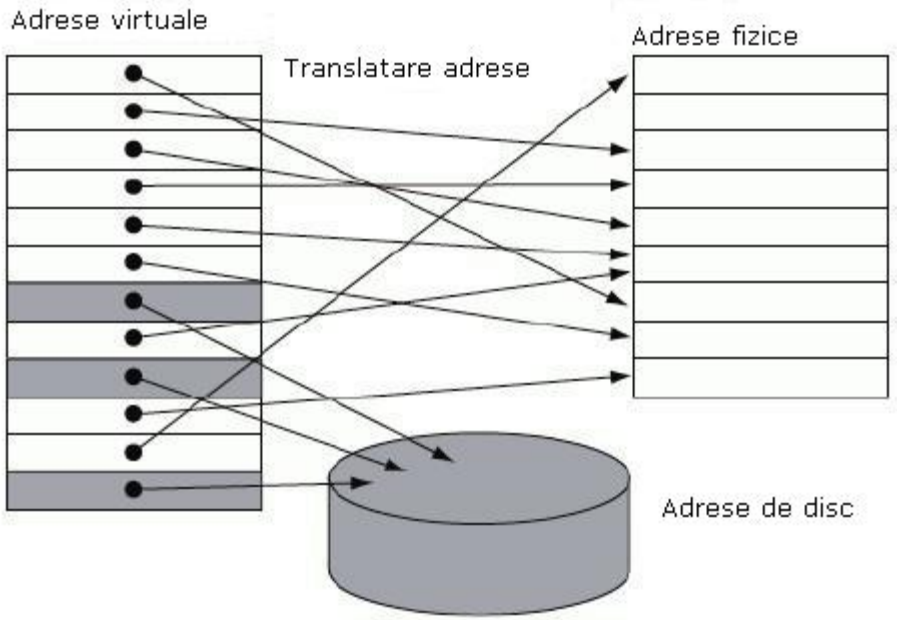
Până la apariţia acestui concept, depăşirea dimensiunii de memorie implica intervenţia programatorului.

Se împărţea programul în componente şi se trecerea la determinarea excluderilor mutuale.
Suprapunerile erau încărcate sau scoase din memorie în timpul execuţiei programului.
Apelurile dintre procedurile aflate în diferite module determinau suprapunerea unui modul cu altul.

Un bloc de memorie virtuală este denumit ***pagină***, iar un eşec la accesarea memoriei virtuale se numeşte ***page fault***.

Pentru memoria virtuală vom avea ***adrese virtuale*** ce sunt translatate în ***adrese fizice***.

***Exemplu***: Adresa virtuală este numele unei cărţi iar adresa fizică reprezintă locaţia cărţii în bibliotecă.




Procesorul generează adrese virtuale în timp ce memoria este accesată folosind adrese fizice

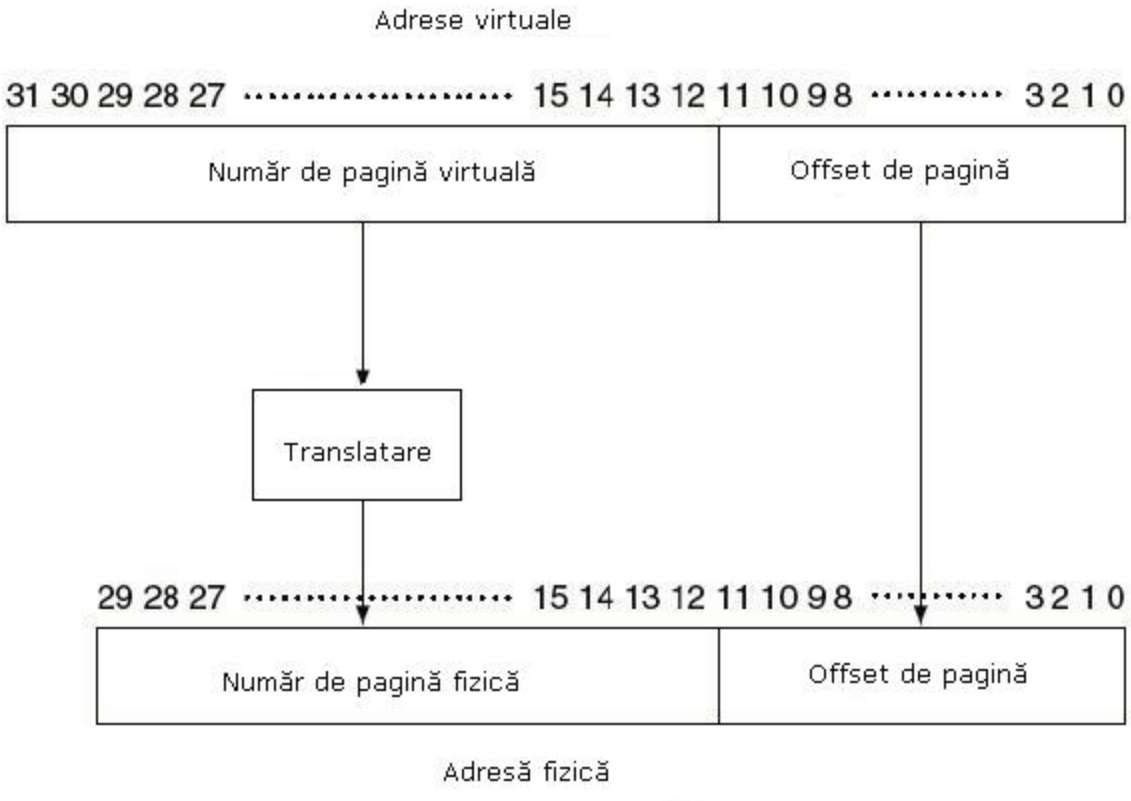
Ambele memorii sunt compuse din pagini (virtuale/fizice) între care există o corespondenţă de 1:1

Există posibilitatea ca o pagină virtuală să fie prezentă numai pe disc => imposibilitatea de a avea ca şi corespondenţă o pagină fizică.

O pagină fizică  poate fi folosită în comun – două adrese virtuale fac referire la aceeaşi adresă fizică.

Memoria virtuală oferă mecanismul de ***realocare*** – se calculează corespondenţa dintre adresele virtuale folosite de program şi diferitele adrese fizice , înainte ca adresele fizice să fie folosite de program.

Realocarea se face pe bază de blocuri de dimensiune fixă.



***Proiectarea sistemelor de memorie virtuală***

Dimensiunea paginilor trebuie să fie mare pentru a amortiza timpul de acces ridicat – 32KB sau 64KB spre exemplu

Amplasarea complet asociativă a paginilor – se reduce complet frecvenţa de page fault.

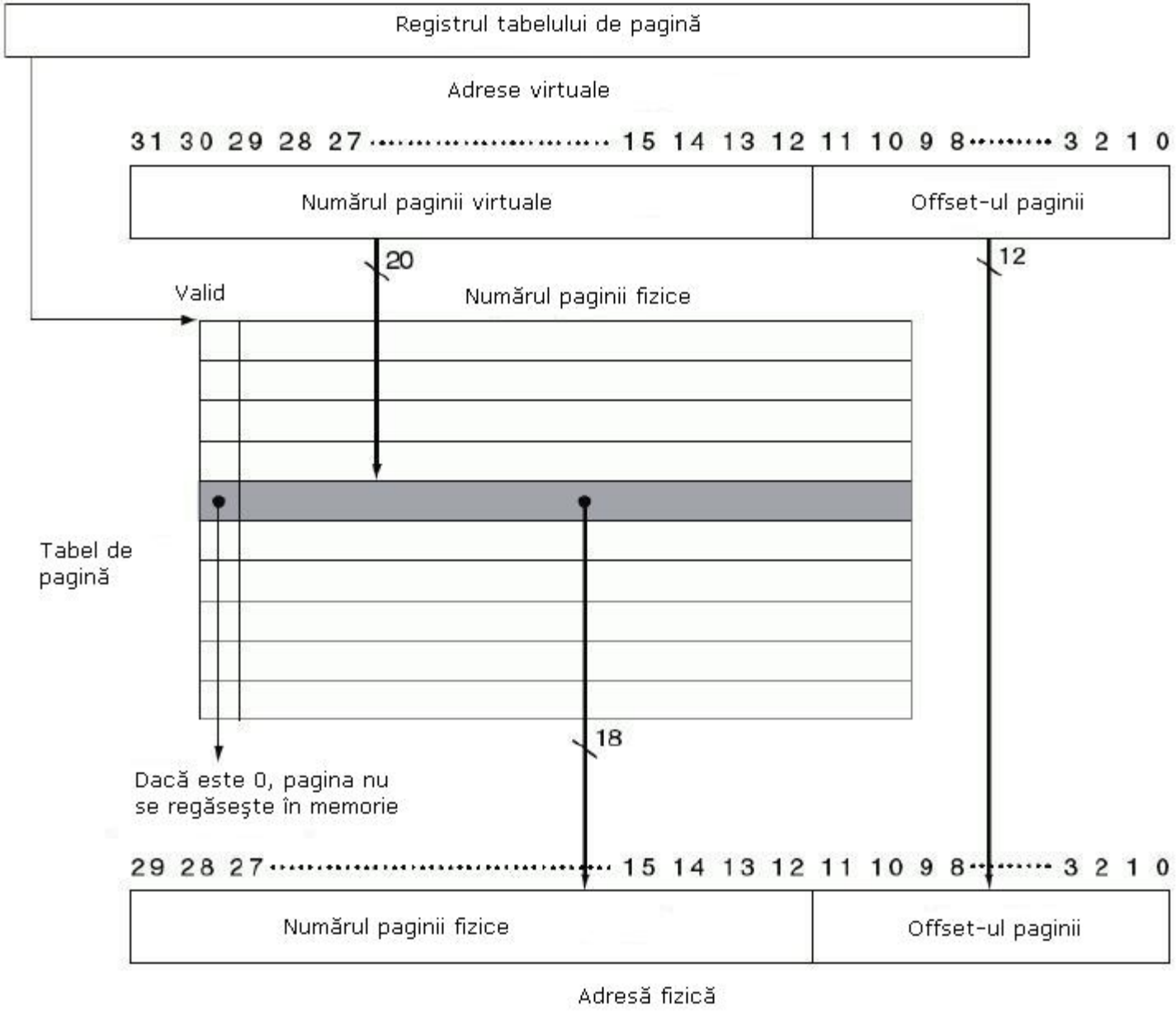
Page fault-urile pot fi tratate prin intermediul software-ului.

În memoria virtuală, paginile sunt localizate prin folosirea unui tabel ce indexează memoria; acestă structură se numeşte ***page table – tabel de pagină***.

Fiecare program are tabelul său de pagini, ce conţine corespondenţa dintre spaţiul său de adrese virtuale şi adresele memoriei principale.

Adresa unui tabel este memorată într-un registru ce indică adresa de început a tabelului – ***page table register***.

!!!! Presupunem că tabelul se găseşte într-o regiune fixă şi continuă din memorie


Registrul tabelului de pagină
Adrese virtuale
31 30 29 28 27 ..... 15 14 13 12 11 10 9 8 ..... 3 2 1 0
Numărul paginii virtuale
Offset-ul paginii
20
12
Valid
Numărul paginii fizice
Tabel de pagină
18
Dacă este 0, pagina nu se regăseşte în memorie
29 28 27 ..... 15 14 13 12 11 10 9 8 ..... 3 2 1 0
Numărul paginii fizice
Offset-ul paginii
Adresă fizică

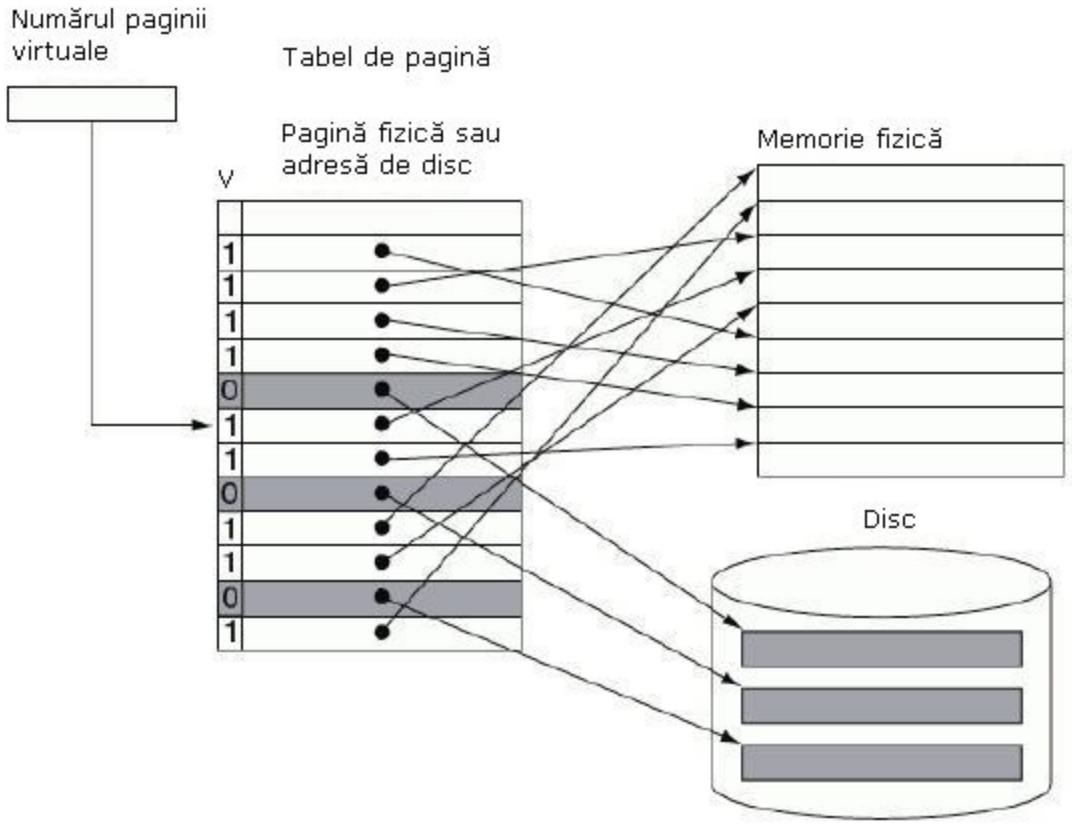
## *Page faults*

În cazul în care bitul de validare este 0 (apare page fault) controlul va fi preluat de SO prin intermediul unui mecanism de tratare a excepţiilor

SO-ul caută pagina în următorul nivel de memorie din ierarhie, adresa virtuală nu poate indica unde se găseşte pagina cerută.

SO-ul creează un spaţiu pe disc pentru toate paginile unui proces odată cu crearea acestuia, împreună cu o structură de date pentru înregistrarea locului unde este memorată fiecare pagină virtuală pe disc.

SO-ul creează de asemenea, o structură de date ce ţine evidenţa proceselor şi adreselor virtuale folosite de către fiecare pagină fizică.




Tabelele cu adresele paginilor fizice şi ale paginilor de pe disc vor fi memorate în două structuri de date separate.

Cantitatea de memoria folosită pentru memorarea tabelor de pagini este mare.

***Tehnici pentru reducerea volumului de memorie ocupat de tabelele de pagini şi minimizarea memoriei principale alocate acestora***

1. Utilizarea unui registru de limitare ce constrânge dimensiunea tabelului de pagini pentru un proces dat.

2. Majoritatea limbajelor necesită 2 porţiuni expandabile – una ce conţine stiva şi cealaltă ce conţine zona de acumulare (HEAP).

Dezavantaj – nu funcţionează bine atunci când spaţiul de adrese este folosit într-un mod discontinuu.

3. Aplicarea unei funcţii de căutare – HASHING – pentru adresa virtuală, astfel încât structura de date ce conţine tabelul de pagini să aibă o dimensiune egală doar cu numărul de pagini fizice existente în memoria principală.

4. Paginarea tabelor de pagini.

5. Utilizarea mai multor niveluri de tabele de pagini.

# I / O

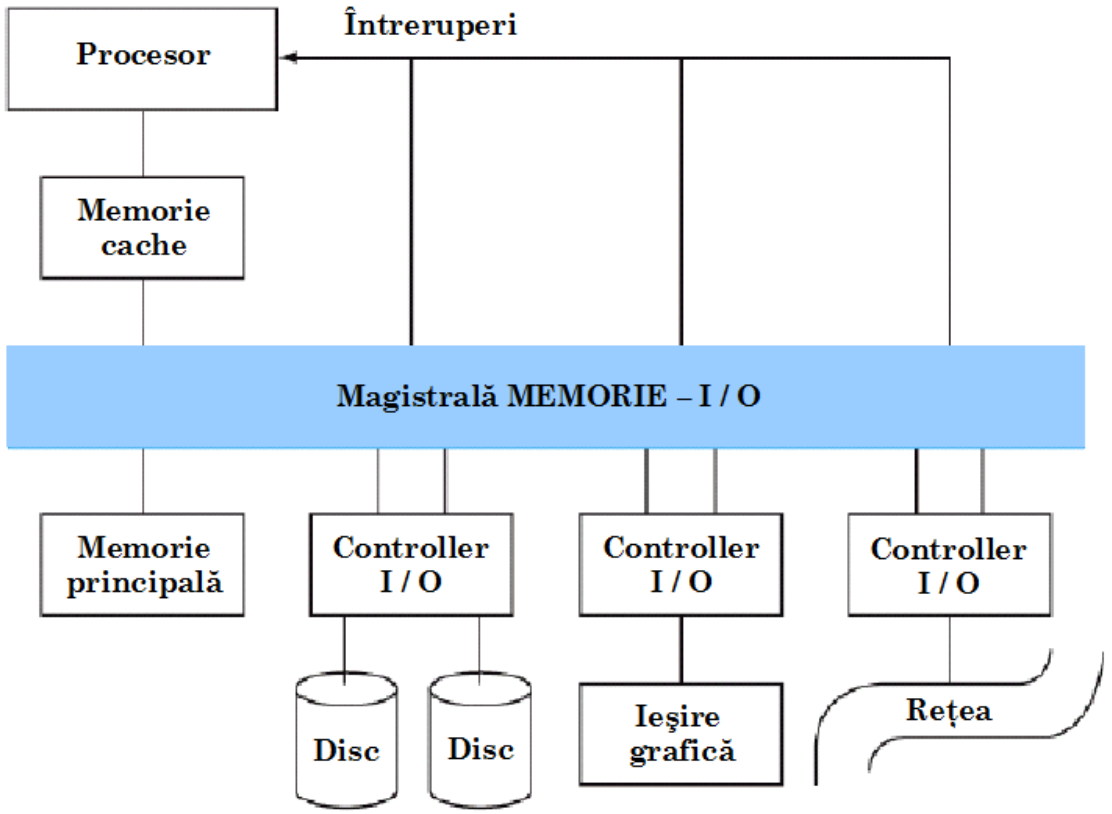
Actualmente există o mare varietate de dispozitive de I/O. Organizarea acestor dispozitive se poate face având în vedere următoarele caracteristici:

1. COMPORTAREA – input (citire o singură dată); output (scriere odată); storage (citire și rescriere)

2. Partenerul – om sau dispozitiv electronic

3. Rata datelor – valoarea maximă cu care datele pot fi transferate între dispozitivul de I/O și memoria principală/procesor

Exp: Tastatura este un dispozitiv de intrare utilizat de către om cu o rată a datelor de peste 10 bytes/sec




Conexiunile între dispozitivele de I/O, procesor și memorie sunt denumite magistrale.
Comunicația între dispozitive și procesor presupune utilizarea întreruperilor precum și folosirea unor protocoale de comunicație.

Performanța I/O depinde de lățimea de bandă existentă între dispozitive. Lățimea de bandă poate fi măsurată prin 2 metode:

1. Cât de multe date pot fi mutate prin sistem într-o anumită perioadă de timp

2. Câte operații de I/O pot fi efectuate într-o unitate de timp

Exemple: În cazul aplicațiilor multimedia lățimea de bandă este folosită în determinarea performanței; în cazul procesărilor unui număr foarte mare de accese al unui dispozitiv I/O, numărul de operații I/O efectuate în unitatea de timp va fi factorul cheie în determinarea performanței.

În cazul calculatoarelor și al laptop-urilor, timpul de răspuns este considerat factorul cheie în determinarea performanței

În cazul dispozitivelor embedded ne interesează durata fiecărui task și numărul de task-uri ce pot fi procesate într/o secundă.

Discurile magnetice – platane rotitoare acoperite cu o suprafață magnetică ce utilizează mișcarea capetelor de citire/scriere pentru accesul la disk.

Este nonvolatil – datele rămân și după întreruperea alimentării cu energie a dispozitivului.

1-4 platane, fiecare având 2 suprafețe ce pot fi scrise;

Stiva de platane este rotită cu o viteză de 5400 – 15000 RPM.

Diametrul platanelor este de la 1 inch la peste 3,5 inch

Fiecare suprafață de disk este împărțită în cercuri concentrice denumite *piste.* Ele sunt în număr de 10000 – 50000 pe o singură suprafață.

Fiecare pistă este împărțită în sectoare (100-500). Fiecare sector are o dimensiune tipică de 512bytes.

Secvența de scriere este următoarea:

- număr sector
- gap
- informația pentru sector + codul de corecție eroare
- gap
- numărul umătorului sector

Inițial toate pistele aveau același număr de sectoare – s-a introdus ZBR (zone bit record)

Capetele de citire/scriere sunt conectate împreună => mișcarea se face în conjuncție.
Fiecare cap este peste aceeași pistă indiferent de suprafață => cilindru.

Accesarea unei date presupune:

*1. Poziționarea capetelor deasupra pistei dorite – seek – seek time*

*2. Se așteaptă până când capetele ajung deasupra sectorului dorit – delay sau rotational delay*

Exp:

*Average rotational latency =* $\frac{0,5}{5400 RPM} = \frac{0,5 \quad rotatii}{5400\ RPM\ /\ (60\ \frac{s}{m})} = 0,0056\ s = 5,6\ ms$

*Average rotational latency =* $\frac{0,5}{15000 RPM} = \frac{0,5 \quad rotatii}{5400\ RPM\ /\ (60\ \frac{s}{m})} = 0,0020\ s = 2,0\ ms$

3. Timpul de transfer – timpul necesar transferării unui bloc de biți.

*Timpul de trasfer = f(dimensiune sector, viteza de rotație, densitatea înregistrărilor a unei piste)*

Controller-ul de disk – are rol de control al discului precum și de control al transferului dintre disk și memorie.

*La timpul de acces la disk se include și timpul necesar operării controller-ului de disk*

Exp: Să se determine timpul mediu de citire/scriere al unui sector de 512bytes pentru un disk care are o viteză de rotație de 10000 RPM. Se cunosc:

*1.Timpul mediu de poziționare dat = 6 ms*
*2.Rata de transfer = 50 MB/s*
*3.Overhead-ul controller-ului este de 0,2 ms*
*4.Presupunem că discul este idle => nu există timp de așteptare*

Soluție

*Timpul mediu de acces al discului = timpul mediu de poziționare + întârzierea medie + timpul de transfer + overhead-ul controller-ului*

$$6{,}0\ ms + \frac{0{,}5\ rot}{10000\ RPM} + \frac{0{,}5\ KB}{50\ MB/s} + 0{,}2\ ms = 6{,}0 + 3{,}0 + 0{,}01 + 0{,}2 = 9{,}2\ ms$$

*Dacă timpul mediu de poziționare măsurat este 25% din timpul mediu dat, atunci avem: 1,5 ms + 3,0 ms + 0,01 ms + 0,2 ms = 4,7 ms*

## RAID – Redundant Arrays of Inexpensive Disks

O organizare de disk-uri care utilizează o matrice de disk-uri mici (ca și capacitate) și ieftine pentru creșterea performanțelor și a siguranței în utilizare.

Ideea a fost de înlocuire a discurilor mari cu disk-uri mici. Disk-urile mici sunt mult mai eficiente per gigabyte decât disk-urile mari (evident ne referim la cantitatea de date stocată pe un astfel de disk)

RAID 0 – nu avem redundanță – striping

Presupune răspândirea datelor pe mai multe disk-uri => acces automat la mai multe disk-uri simultan. Din punct de vedere al utilizatorului există doar un singur disk, ceea ce simplifică managementul informației.

Performanță mare pentru accese la informație de dimensiune mare (sisteme de editare video) deoarece mai multe disk-uri funcționează ca unul singur.

RAID 1 – toleranță la defecte – mirroring sau shadowing

Este modalitatea aleasă atunci când toleranța la defecte este un punct critic. Numărul de hardisk-uri utilizat este dublu față de RAID 0.

Când o dată este scrisă pe un disk, automat datele sunt scrise pe un alt disk redundant => întotdeauna vom avea 2 copii ale informației. Dacă un disk va prezenta un defect la un moment dat, atunci informația va fi citită de pe discul oglindă. CEA MAI SCUMPĂ SOLȚIE RAID.

RAID 2 – detectare și corectare erori

Împrumută tehnicile de detecție și corecție a erorilor folosite în cazul memoriilor

RAID 3 – grup de protecție – bit-interleaved parity

Costul unei disponibilități mărite a datelor poate fi redus la 1/N, unde N este numărul de disk-uri care fac parte dintr-un grup de protecție.

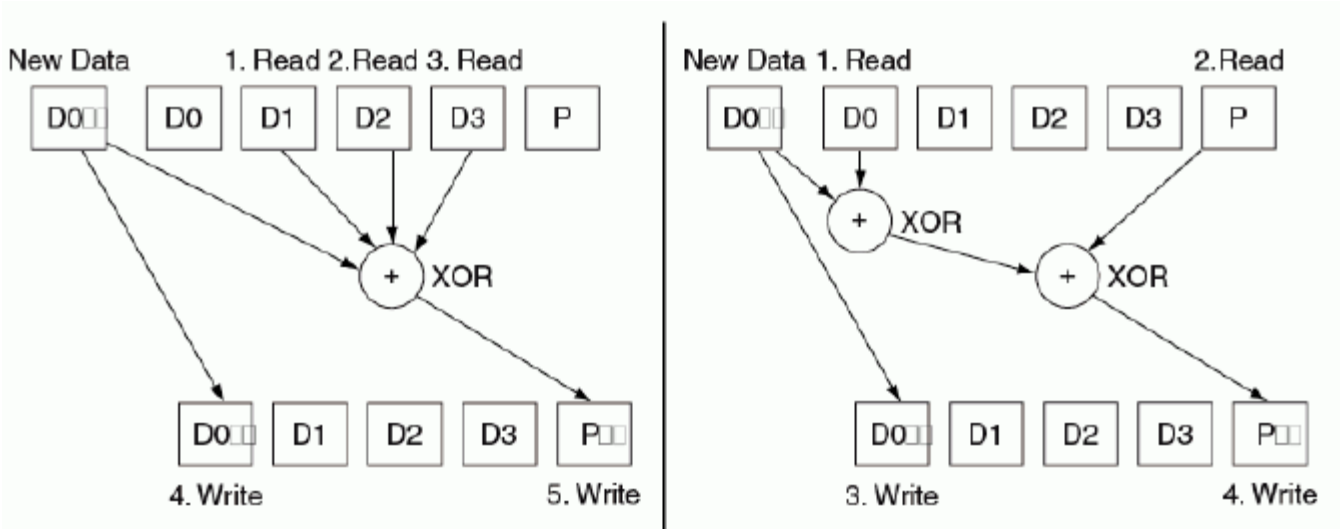
Decât să adăugăm disk-uri, mai simplu ar fi să adăugăm informație redundantă pentru restaurarea informației pierdute în caz de crash.

Citirile/scrierile se fac pe toate disk-urile din grup, dar vom avea 1 extra disk pentru menținerea informației de verificare în caz de crash. Schema folosită este determinarea parității informației.

RAID 3 este foarte folosit în cazul aplicațiilor care utilizează seturi de date foarte mari – multimedia sau cod științific

RAID 4 – block-interleaved parity

Similar cu RAID 3, doar că utilizează un acces al datelor diferit. Paritatea este memorată ca blocuri și asociată cu un set de blocuri de date.



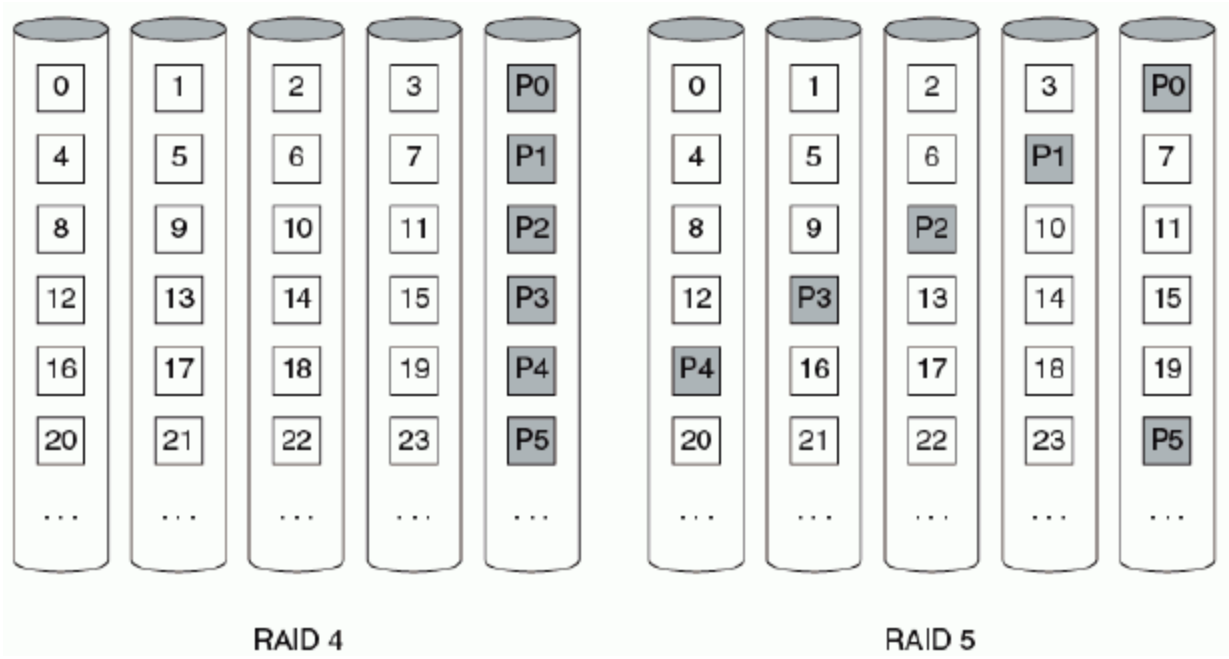

RAID 3 vs RAID 4

Sunt optimizate scrierile mici, deci vom avea un număr redus de accese la disk precum și un număr mic de disk-uri ocupate.

## RAID 5 – distributed block-interleaved parity

Dezavantajul lui RAID 4 este faptul că paritatea disk-ului trebuie recalculată la fiecare scriere.

O soluție ar fi să distribuim această informație pe toate discurile astfel încât să nu mai avem un singur bottleneck.




Prin această soluție, anumite scrieri mici pot fi executate în paralel.

## RAID 6 – P + Q redundancy

Folosit în cazul în care o singură corecție nu este suficientă. Putem generaliza paritatea pentru a avea o nouă calculație pentru date și o nouă informație pentru verificarea discului.

Acest block secundar este folosit pentru recuperarea datelor în caz de eșec multiplu. Overhead-ul este dublu față de RAID 5.

Alte metode folosite în practică

1.HOT SWAPPING – replasarea unei componente hardware cât timp sistemul este în stare de funcționare

2.STANDBY SPARES – Cuplarea unor resurse hardware noi imediat ce o resursă hardware este defectă

# MAGISTRALE - BUSES

O magistrală conține un set de linii de control și un set de linii de date.

Liniile de control sunt utilizate pentru semnale de tip cerere și confirmare și ele indică tipul informației existentă pe liniile de date.

Liile de date sunt folosite la transportarea informației între sursă și destinație. Acestă informație poate să conțină comenzi complexe, date și adrese.

Bus transaction – o secvență de operații care include o cerere și poate include un răspuns. O tranzacție este inițiată de un singur request și poate avea mai multe operații individuale de magistrală.

Processor-memory bus

Backplane Bus - o magistrală care este proiectată pentru a permite coexistarea pe o singură magistrală a următoarelor componente: procesor, echipamente I/O, memoria

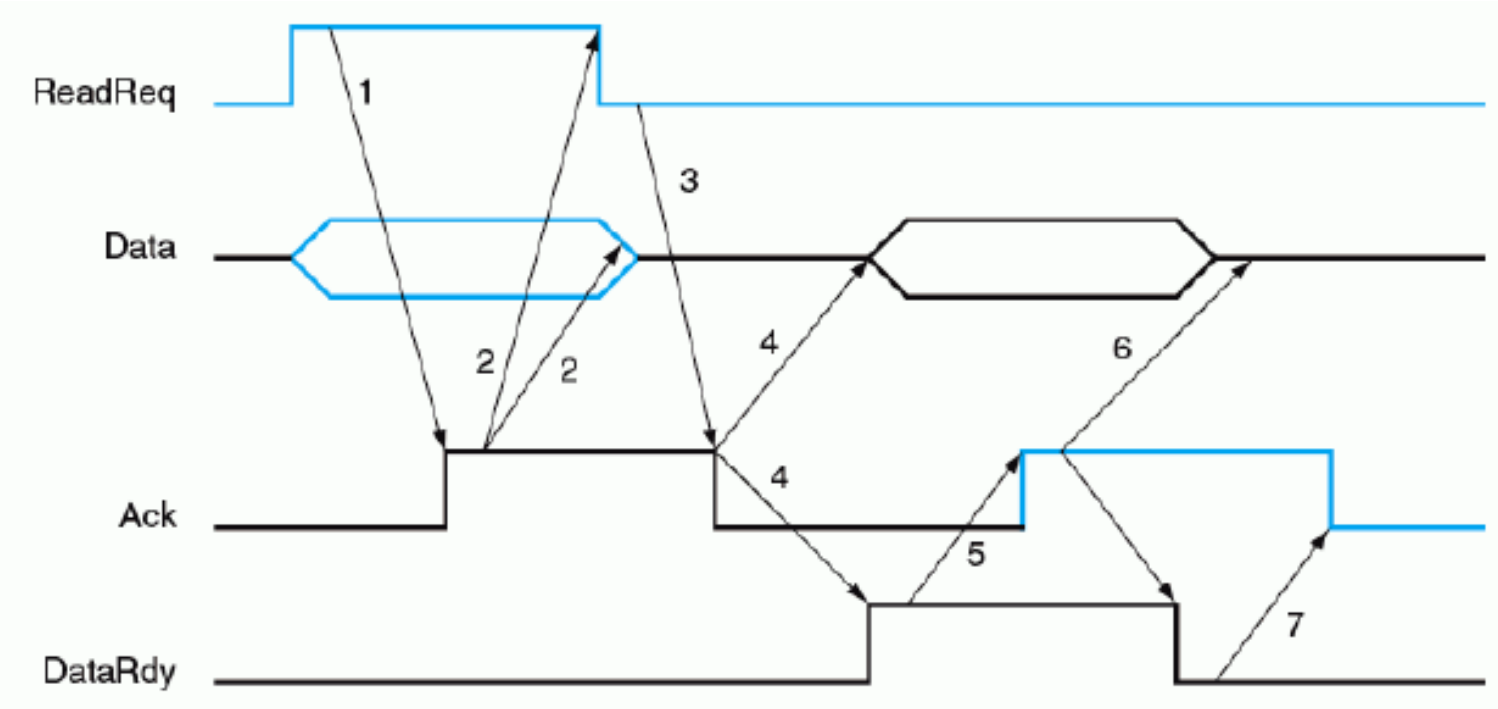
Magistrală sincronă – O magistrală care include ceasul în liniile de control și un protocol fix pentru comunicarea relativă la frontul de ceas.

Magistrală asincronă – O magistrală care utilizează protocolul HANDSHAKING pentru coordonarea utilizării în locul ceasului. Este folosită ca o punte de legătură între dispozitive care operează la viteze diferite

| Bus type | I/O | I/O |
|---|---|---|
| Basic data bus width (signals) | 4 | 2 |
| Clocking | asynchronous | asynchronous |
| Theoretical peak bandwidth | 50 MB/sec (Firewire 400) or 100 MB/sec (Firewire 800) | 0.2 MB/sec (low speed), 1.5 MB/sec (full speed), or 60 MB/sec (high speed) |
| Hot plugable | yes | yes |
| Maximum number of devices | 63 | 127 |
| Maximum bus length (copper wire) | 4.5 meters | 5 meters |
| Standard name | IEEE 1394, 1394b | USE Implementors Forum |

Firewire (1394) USB 2.0

# Protocolul - HANDSHAKING




Split Transaction Protocol – Un protocol în care magistrala este eliberată pe durata unei transacții de magistrală cât timp cel care a lansta cererea este în așteptare pentru datele ce vor fi transmise.

# Studiu de caz – PENTIUM 4