Curs 1 (Concepte de baza) :

Time-sharing SO : MIT CTSS ’62, MULTICS ’65, UNIX ‘69

Multi-processing SO : arhitecturi SMP = UNIX, Linux, Windows, etc.

Clasificarea SO-urilor :
1) dupa nr taskuri:

- mono-tasking : CP/M, DOS
- multi-tasking : UNIX, OS/2, Windows 3.x/9x, Windows NT

2) dupa nr de utilizatrori deserviti simultan :

- mono-utilizator : CP/M, DOS, Macintosh, OS/2, Windows Desktop
- multi-utilizator : UNIX, Mac OS X, Solaris, Windows server versions (NT)

Curs 2 (Organizare / Nucleu) :

Termeni I/E :

- Trap = o intrerupere generata software, cauzata de o eroare sau cerere explicita dintr-un program utilizator
- Polling = activitatea de determinare a tipului de intrerupere aparuta

Operarea in mod dual a CPU-ului :

- User-mode = neprivilegiat
- Kernel-mode = privilegiat

Neprivilegiat -> Privilegiat : SYSCALL/SYSRET (amd64 sau x86 amd) sau SYSENTER/SYSEXIT (x86 Intel)

Nucleul -> cea mai importanta parte a SO-ului; Ofera servicii pentru procesele utilizator prin intermediul apelurilor de sistem; Gestioneaza procesele, memoria, sistemele de fisiere si perifericele I/E.

Tipuri :

Monolitic (UNIX, VMS, DOS, Windows 3.x/9x)

Monolitic Modular (Linux)

Micro-nucleu (Maxh, L4, MINIX, QNX)

Hibrid (NT Kernel : Windows NT4/2000/XP/Vista/Win7/8/10 & Win Srv, ReactOS)

Curs 3 (Gestiunea proceselor I) :

Starile procesului : running / waiting / ready. La sistemele uniprocesor, in orice moment, un singur proces poate fi “running”.

Proces -> independent = daca nu poate afecta si nici nu poate fi afectat de alte procese ce se executa in sistem

Proces -> cooperant = poate (fi) afecta(t) /de alte procese

Procese multiple pot fi multiprogramate pe un singur CPU (i.e., executate prin “paralelism aparent”)

Procese concurente :

- Ce se întâmplă dacă fiul face exit (se termină) înainte ca tatăl să facă join?

  Un obiect proces “zombie” păstrează codul de terminare șiinformațiile de stare ale fiului

- Ce se întâmplă dacă tatăl se termină înaintea fiului?

  Orfanii devin copii ai procesului init (cu PID-ul 1)

- Ce se întâmplă dacă tatăl nu-și poate permite să aștepte la un punct de join?

  Facilități pentru notificări asincrone (prin semnale Unix)

Activitatea SO-ului -> la 3 nivele :

1) Nivelul inalt (planificarea joburilor) : se decide care joburi pot intra in sistem pentru a concura pt resursele acestuia
2) Nivelul de mijloc (planificarea proceselor) : se ajusteaza prioritatile proceselor si se pot suspenda procese, determinand astfel care procese vor concura pt CPU
3) Nivelul scazut (dispecerat) : se decide carui thread I se va aloca efectiv CPU-ul

Planificatoare :

1) Pe termen lung (planificator de joburi) : selecteaza procesele si le incarca in memorie pentru executie; controleaza gradul de multi-programare (nr de procese din memorie)
2) Pe termen scurt (planificator CPU) : selecteaza dintre procesele ready, unul caruia ii aloca CPU-ul pentru urmatoarea cuanta de timp procesor ; acest planificator trebuie sa fie foarte rapid, deoarece va fi executat cel putin o data la (aprox) 10ms

*Pt sistemele moderne, planificatorul pe termen lung poate fi minimal/absent.

Curs 4 (Gestiunea proceselor II) :

Deciziile de planificare a CPU se iau in urmatoarele situatii :

1. Cand un proces/thread trece din starea running in starea waiting
2. Cand un proces/thread trece din starea running in starea ready
3. Cand un proces/thread trece din starea waiting in starea ready
4. Cand executia unui proces/thread se termina (normal sau fortat)

Alg preemptivi (SRTF, cu prioritati preemptiva, RR)

Non-preemptivi (FCFS, SJF nepreemptiv, cu prioritati nepreemptiva)

Tranzitie de la running la ready -> algoritmi preemptivi

Algoritmi de planificare :

1) FCFS : Procesul care solicita primul sa i se acorde timp CPU, este primul caruia i se va aloca CPU-ul ; Algoritm ne-preemptiv (nu poate fi utilizat pentru medii interactive)
2) SJF : scoaterea rapidă din sistem a proceselor scurte pentru a minimiza numărul de procese aflate în așteptare cât timp rulează un proces lung ; Planificare ce poate fi ne-preemptiva sau preemptiva
3) SRTF (SJF preemptiv)
4) Planificarea cu prioritati : CPU se aloca procesului cu cea mai mare prioritate, in caz de egalitate = FCFS
5) Round-Robin : Fiecarui proces I se ofera CPU pentru o cuanta de timp

Curs 5 (Sincronizarea proceselor I) :

Problema Sectiunii Critice :

Pt n=2 -> Peterson ’81 (se partajeaza turn si flag[]), prima solutie = Dekker ‘65

n>2 -> Algoritmul brutarului. Prima solutie corecta : Eisenberg & McGuire ‘72

Solutie hardware completa = Test-and-Set

Solutii concrete : Lacate mutex (spinlocks) ; Semafoare (Dijkstra ’65)

Deadlock (interblocaj) = O situație în care două sau mai multe procese așteaptă pe termen nelimitat producerea câte unui eveniment (e.g., execuția unei operații signal pe un semafor), eveniment ce ar putea fi cauzat doar de către unul dintre celelalte procese ce așteaptă.

Starvation (blocaj nelimitat) = O situație în care un(ele) proces(e) așteaptă nelimitat (e.g., la un semafor: procesul ar putea sta suspendat în coada de așteptare asociată acelui semafor pe termen nelimitat).

Curs 6 (Sincronizarea proceselor II) :

Problema Producator-Consumator : Reprezentativa pentru ilustrarea conceptului de procese cooperante. Solutia (buffer nelimitat) = un semafor binar mutex (controleaza accesul la SC) ; un semafor binar delay (blocheaza consumatorul daca buffer gol ; un n care numara elementele din buffer. Init n=0, mutex=1, delay=0. Solutia (buffel limitat) = se adauga semafor binar delayPro (blocheaza producatorul daca buffer este plina) | SAU | : un semafor binar mutex ; un semafor general empty (numara locatiile goale din buffer) ; un semafor general full (numara locatiile pline din buffer). Init mutex=1, full=0, empty=n.

Problema Cititori si Scriitori (CREW) : Reprezentativa pentru accesul la o baza de date. Crew = Este acceptabil sa avem mai multe procese care sa citeasca dar cand un proces scrie, nici un alt proces nu trebuie sa aiba acces la ea, nici macar de citire.

Problema Cina Filozofilor Chinezi : Reprezentativa pentru nevoia de a aloca un numar limitat de resurse (nepartajabile) la mai multe procese, ce concureaza pentru acces exclusiv la aceste resurse, alocare care sa se faca fara sa apara fenomenul de interblocaj sau cel de infometare.

Problema Barbierului Adormit : Reprezentativa pentru situatii diverse de asteptare la coada

Monitorul -> Concept dezvoltat de B.Hansen ’73 & C.A.R.Hoare ’74. Este o constructie de sincronizare de nivel inalt = se elimina erorile de programare ce pot aparea la folosirea semafoarelor datorita unei ordini incorecte a apelurilor wait si signal.

Curs 7 (Comunicarea inter-procese / Fenomenul de interblocaj) :

2 modele de mecanisme care sa permita comunicatia intre procesee ce doresc sa coopereze :

1) Sisteme cu memorie partajata (shared memory) – necesita ca procesele comunicante sa partajeze niste variabile (zone) de memorie
2) Sisteme cu mesagerie (message-passing) – permit proceselor sa schimbe mesaje intre ele. Primitivele sistemelor cu mesaje = send(mesaj), receive(mesaj).

IPC sub UNIX :

- pipe-uri (canale anonime) : pipe() ; pot fi utilizare doar de procese inrudite prin fork()
- named pipe-uri (canale cu nume) : mkfifo() ; pot implementa schimbul de mesaje intre procese oarecare, dar locale
- socket-uri : pot implementa schimbul de mesaje intre procese oarecare, la distanta
- semnale UNIX
- semafoare (UNIX System V) : implementare cu mesaje;
  semget()/semop()/semctl() = UNIX System V ;
  creatsem()/opensem()/waitsem()/sigsem() = Xenix
- zone de memorie partajata (UNIX System V) :
  shmget()/shmat()/shmdt()/shmctl()
- cozi de mesaje (UNIX Sytem V) : msgget()/msgsnd()/msgrcv()/msgctl()

Alte mecanisme de IPC (UNIX/Windows) :
RPC (Remote Procedure Calls) si RMI (RPC in Java)

Standardul MPI (Message Passing Interface)

Interblocajul proceselor := doua sau mai multe procese asteapta la infinit producerea unor evenimente ce pot fi cauzate doar de unul sau mai multe dintre procesele ce asteapta, prin urmare nici un evenimen nu se va produce niciodata.

Sunt necesare 4 conditii pentru a fi posibila aparitia interblocajului :

1) excluderea mutuala
2) hold & wait
3) no preemption
4) asteptare circulara

RAG (graful de alocare a resurselor) = G graf bipartit orientat (V,E)

Varfurile din V sunt elemente de doua tipuri :

- P = {P1, P2, …, Pn} : toate procesele din sistem
- R = {R1, R2, …, Rm} : toate resursele din sistem

Arcele din E sunt de doua tipuri :

- Arce de cerere : arc orientat Pi -> Rj
- Arce de alocare : arc orientat Rj -> Pi

Obs:

- Daca este interblocaj -> Exista circuit in RAG
- Daca nu exista circuite in graful de alocare a resurselor, atunci nu exista interblocaj
- Daca, in schimb, exista un circuit in RAG, aceasta in general este o conditie necesara pentru interblocaj, dar nu este si suficienta

Solutii Interblocaj :

- Ignorare -> cele mai multe SO-uri ignora interblocajele (sistemul se va comporta “ciudat”, moment la care va putea fi rebootat)
- Prevenire -> Presupune eleminare a cel putin uneia dintre cele 4 conditii necesare pentru interblocaj, facand astfel interblocajul imposibil :
  - Excluderea mutuala -> “Relaxarea” ei ar insemna folosirea simultana, de catre mai multe procese, a unei resurse. Ar fi posibila doar pentru resurse partajabile, nu si pentru cele exclusive gen imprimanta.
  - Hold & Wait -> “Relaxarea” ei ar insemna Sa nu lase procesul sa pastreze resursele in timpul cat asteapta sa obtina alte resurse
  - Ne-preemptie -> “Relaxarea” ei ar insemna preemptie, adica sa permita SO-ului sa ia inapoi resursele de la acele procese ce detin

resurse, dar asteapta pentru alte resurse ; Resursele luate sunt adaugate la lista resurselor de care procesul “victima” are nevoie pentru a-si continua executia.
- Asteptarea circulara -> “Relaxarea” ei ar putea fi impunerea unei ordonari a resurselor.

❖ Evitare -> Foloseste alocarea controlata : se examineaza toate cererile de alocare a resurselor si se iau decizii astfel incat sa se impiedice aparitia interblocajului. == Algoritmul Bancherului
❖ Detectie -> Strangerea unui RAG intr-un graf wait-for (nu functioneaza In cazul resurselor cu instante multiple)

Infometare := situatia care apare atunci cand un proces, ce a cerut permisiunea de acces la o resursa asteapta la infinit primirea acelei resurse, deoarece exista un flux constant de alte procese care solicita acea resursa si o si primesc, in defavoarea procesului infometat.

Solutii infometare :

❖ Ignorare : o lasam in sarcina operatorului uman
❖ FIFO : implementarea unei politici de servire care sa respecte ordinea de primire a cererilor (e.g. la alocarea CPU-ului FCFS sau RR, sau pt semafoare implementarea cozii de asteptare sub forma de coada FIFO)
❖ Aging-ul : implementarea unei politici de servire care sa favorizeze procesele ce asteapta de mult timp primirea resursei solicitate (e.g. la alocarea CPU-ului alg de planificare cu prioritati dinamice, in care prioritatea unui proces este marita treptat in timp ce este ready, cu revenirea la valoarea initiala a prioritatii dupa ce ruleaza o cuanta)

Curs 8 (Administrarea memoriei I) :

Ierarhii de memorii : memorii cu acces mai rapid, dar realizate în tehnologie mai costisitoare și deci de dimensiune mai mică // memorii cu acces mai lent, dar realizate în tehnologie mai ieftină și deci de dimensiune mai mare

Memorii interne : memoria principala RAM si memoriile cache din CPU

Memorii externe : memoria secundara de stocare (HDD/SSD) si memoria tertiara de arhivare (discuri externe, CD/DVD-uri, stick-uri USB, …)

Astfel se realizeaza un mecanism, transparent pentru utilizator, prin care SC-ul lucreaza cu o cantitate mica de memorie rapida si una mare de memorie lenta, creand aparenta ca ar avea numai memorie rapida si in cantitate mare.

Legarea adreselor (address binding) : se poate face la

1) Momentul compilarii – se genereaza cod executabil cu adrese fixe. Programul trebuie recompilat daca codul trebuie incarcat la o alta adresa de memorie (DOS fisiere .com)
2) Momentul incarcarii – se genereaza adresele atunci cand programul este incarcat in memorie (DOS fisiere .exe)
3) Moementul executiei (dinamic) – adresele sunt relative la o valoare care se poate modifica pe parcursul executiei, i.e. cod relocabil la runtime (Windows fisiere .exe)

Alocarea memoriei :

- Contigua vs necontigua :
  - Contigua : pentru un proces se aloca o singura portiune, continua, din memoria fizica
  - Necontigura : pentru un proces se aloca mai multe portiuni separate din memoria fizica
- Reala vs virtuala :
  - Memoria reala : spatiul de adresare al proceselor este limitat de capacitatea memoriei interne
  - Memoria virtuala : spatiul de adresare nu este limitat de capacitatea memoriei interne (suplimentata de cea externa prin swapping)

Alocarea dinamica (in partitii variabile) : algoritmi :

- FFA : se parcurge lista spațiilor libere (ordonată crescător după adresa de început a spațiilor) și se alege primul spațiu de dimensiune suficientă (e.g. MINIX)

- BFA : se parcurge lista spațiilor libere (ordonată crescător după dimensiunea spațiilor) și se alege primul spațiu de dimensiune suficientă (e.g. DOS). Produce cel mai mic spațiu liber rest.
- WFA : se parcurge lista spațiilor libere (ordonată crescător după dimensiunea spațiilor) și se alege ultimul spațiu din listă. Produce cel mai mare spațiu liber rest.

In general FFA si BFA sunt mai buni decat WFA, dar in anumite scenarii oricare poate fi mai bun decat ceilalti doi.

Sufera de fragmentare interna:
- Alocarea (contigua) in partitii fixe
- Paginarea

Sufera de fragmentare externa:
- Alocarea (contigua) in partitii variabile
- Segmentarea

Curs 9 (Administrarea memoriei II) :

Alocarea paginata :

● memoria fizică este împărțită în blocuri de lungime fixă, numite cadre de pagină (frames), sau pagini fizice

● lungimea cadrelor de pagină este o putere a lui 2 și este o constantă a SC-ului (e.g. pentru arhitecturile Intel este 4 Ko)

● memoria logică a unui proces (zona de cod + zona de date) este împărțită în blocuri de lungime fixă (egală cu lungimea cadrelor de pagină), numite pagini (pages), sau pagini virtuale

Tabela de pagini in memorie :

– O tabelă de pagini per proces

– Un registru, PTBR (page table base register), conține adresa de bază (i.e., de început) în memorie a tabelei de pagini (e.g. registrul CR3 în cazul CPU x86)

– Un alt registru, PTLR (page table length register), indică dimensiunea tabelei de pagini

– Problemă: fiecare acces la date sau instrucțiuni “costă” două accese de memorie

– Soluția: TLB (traslation look-aside buffer)

TLB -> Calitate fundamentala = adresarea prin continut, si nu prin adresa : gaseste locatia care are un continut specificat, cautand simultan (in paralel) in toate locatiile ei.

TLB are rol de cache pentru tabela de pagini din memorie: când se încearcă aflarea numărului de cadru asociat unui număr de pagină virtuală, aceasta este căutată mai întâi în TLB, iar dacă nu-l găsită în TLB, este căutată în tabela de pagini din memorie

Performanta adusa de TLB este data de Hit_ratio = procentajul de gasiri in TLB a paginii cautate ;

Timpul efectiv de acces [ar trebui pe ambele arhitecturi] : EAT =

1*Memory_AT + (Hit_ratio * TLB_AT + (1-Hit_ratio) * (TLB_AT + Memory_AT))

Valori uzuale : Memory_AT >> TLB_AT si Hit_ratio aproximativ 90-95%. Ca urmare… EAT este foarte apropiat de 1*Memory_AT.

Segmentarea = “paginare” in segmente ; Este o schemă de administrare a memoriei care suportă vederea utilizatorului asupra memoriei – programul este divizat în mai multe unități logice

Segmentarea paginata : Introdusa de SO-ul MULTICS ; Fiecare segment este impartit in pagini -> Se elimina alocarea contigua si fragmentarea externa.

Memoria virtuala : ideea de Swapping. Principiul localitatii = doua tipuri de localizare a secventelor de accesare a memoriei :

1) Localitate temporala – tendinta de a accesa in viitorul apropiat locatii accesate deja in istoria recenta
2) Localitate spatiala – tendinta de a accesa in viitorul apropiat locatii apropiate de alte locatii accewsate deja in istoria recenta.

Curs 10 (Administrarea memoriei III) :

Paginarea la cerere = paginarea combinata cu tehnica de swapping.

Ideea : aducerea paginilor pe disc in memorie doar in momentul cand sunt referite (“cand e nevoie de ele”) -> Se elimina restrictia ca programul sa fie prezent in intregime in memorie pentru a putea fi executat.

Trashing = fenomenul de sufocare a SC-ului cu rezolvarea erorilor de pagina, in loc de a-si folosi resursele pentru executia job-urilor utile.

– Cum împiedicăm utilizatorii să acceseze datele protejate?

● drepturi de acces specificate la nivel de pagină

– Dacă o pagină este prezentă în memorie, cum o găsim?

● tabela de mapare a paginilor (în memoria fizică)

– Dacă o pagină nu este prezentă în memorie, cum o găsim?

● tabela de mapare (a paginilor) pe disc

– Când este adusă o pagină în memorie?

● politici de încărcare – la cerere (atunci când este nevoie de ea)

– Dacă o pagină este adusă în memorie, unde o punem?

● politici de plasare

– Dacă o pagină este evacuată din memorie, unde o punem? Cum decidem care pagini să fie evacuate din memorie?

● politici de înlocuire – algoritmi de page swapping

Suportul hardware pentru paginare la cerere : MMU (Memory Management Unit, located in CPU) :

SO-ul controleaza MMU pentru a selecta :

1) Submultimea de posibile adrese virtuale ce sunt valide pentru fiecare proces
2) Translatarile fizice pentru acele adrese virtuale
3) Modurile de acces permis la acele adrese virtuale
4) Setul specific de translatari valabil la un moment dat

Suportul software pentru paginare la cerere : Algoritmi de inlocuire a paginilor :

- Algoritm de inlocuire a unei pagini neutilizate recent : NRU
- Algoritm de inlocuire in ordinea incarcarii : FIFO
- Algoritm de inlocuire a celei mai putin utilizate recent pagini accesate in trecutul recent: LRU
- Algoritm de inlocuire a celei mai putin utilizate recent pagini in trecut: LFU
- Aproximari LRU pentru implementari
- Alte abordari
- Implementari reale

ALGORITMUL NRU (= Not Recently Used)

- Ideea : inlocuirea unei pagini care nu a fost utilizata recent
- Fiecare pagina fizica are asociati doi biti in tabela de paginare, folositi pentru a decide ce pagina se va evacua
- Bitul de referire este resetat (pus pe 0) la incarcarea paginii si este setat (pus pe 1) la fiecare acces (referire) la pagina
- Periodic (de obicei 20ms) se face operatia de clearing bits, prin care toti bitii de referire sunt resetati
- Al doilea bit este bitul de modificare (bitul dirty), ce este resetat la incarcarea paginii si setat la scrierea in pagina
- Paginile se impart la fiecare moment in patru clase pe baza acestor biti :
    - Clasa 0 – bitii=(0,0)
    - Clasa 1 – bitii=(0,1)
    - Clasa 2 – bitii=(1,0)
    - Clasa 3 – bitii=(1,1)
- Pagina “victima” se alege din prima clasa nevida (se cauta intai in clasa 0, apoi 1, …); daca pagina aleasa este din clasa 1 sau 3, ea va fi salvata pe disc inainte de a fi inlocuita.

ALGORITMUL FIFO (= First In First Out)

- Ideea : inlocuirea in ordinea incarcarii paginilor
- Implementarea este simpla : se gestioneaza o lista FIFO cu paginile rezidente in memorie; actualizarea ei se face la fiecare incarcare de pagina
- Alegerea “victimei” : prima pagina din lista (cea mai veche)
- Se foloseste si aici bitul dirty (Pentru a se sti daca pagina aleasa trebuie salvata pe disc inainte de inlocuire)
- Anomalia lui Belady : deși bunul simț ne spune că șansa ca o pagină să fie înlocuită scade pe măsură ce crește numărul de cadre fizice, totuși nu se întâmplă așa în cazul alg. FIFO, după cum rezultă și din exemplul anterior

ALGORITMUL LRU (= Least Recently Used)

- Ideea : inlocuirea paginii cel mai putin folosite in ultimul timp
- Cum ? Se va folosi localitatea temporala/spatiala a programului pentru a inlocui pagina cel mai putin utilizata recent
- Nu prezinta anomalia lui Belady
- Spre deosebire de FIFO, alg. LRU avantajeaza paginile “esentiale”, ce vor fi pastrate in memorie

Curs 11 (Administrarea perifericelor de stocare) :

Clasificarea perifericelor de stocare dupa variatia timpului de acces (t_{ij} = timpul de acces de la informatia i la informatia j) :

- Periferice cu acces secvential – t_{ij} are variatii foarte mari (banda magnetica)
- Periferice cu acces complet direct – t_{ij} constant (RAM)
- Periferice cu acces direct – t_{ij} are variatii foarte mici (discul magnetic)

HDDs : Discurile magnetice sunt adresate ca o matrice 1-dimensionala foarte mare de blocuri logice.

Viteza discului :

- Timpul de cautare (seek time) = Timpul necesar pentru miscarea (mecanica) a ansamblului cu capetele de citire-scriere pana la pista specificata
- Latenta de rotatie (rotational Latency) = Timpul de asteptare necesar pentru ca inceputul sectorului specificat sa ajunga prin rotatie sub capul de citire-scriere
- Timpul de transfer (Transfer time) = Timpul necesar pentru a citi datele de pe sectorul specificat
- Timpul total de acces = S + L + T ;

Algoritmi de planificare (a acceselor la disc) : FCFS, SSTF, SCAN, C-SCAN, LOOK, C-LOOK

In SO-urile mai pretentioase se urmareste si minimizarea latentei de rotatie -> Ideea = reordonarea servirilor cererilor existente la un moment dat pentru acelasi cilindru – alg. SLTF (Shortest Latency Time First)

Alocarea blocurilor : o abordare = Grupurile de cilindri utilizate de FFS -> FFS defineste grupurile de cilindri drept unitatea de localitate a discului si factorizeaza localitatea in posibilitati de alegere pentru alocare. Strategia = plasarea blocurilor de date “inrudite” in acelasi grup de cilindri ori de cate ori acest lucru este posibil.

Construirea unui disc mai bun -> SOLUTIE = RAID. Discurile multiple asigura siguranta pastrarii datelor prin redundanta datelor

- Discurile RAID se clasifica in 7 nivele RAID
- Striping-ul utilizeaza un grup de discuri ca o singura unitate de stocare
- Schemele RAID imbunatatesc performanta si siguranta sistemului de stocare prin stocarea redundanta a datelor :
    - Prin tehnica oglindirii (mirroring/shadowing) se pastreaza un duplicat al fiecarui disc
    - Tehnica block interleaved parity -> foloseste mai putina informatie pentru redundanta

NIVELE RAID :

- ♠ Nivelul 0 : fara redundanta, doar striping
- ♠ Nivelul 1 : discuri oglindite
- ♠ Nivelul 2 : coduri Hamming corectoare de erori
- ♠ Nivelul 3 : un disc de paritate la fiecare grup, bit-interleaved
- ♠ Nivelul 4 : citiri/scrieri independente, block-interleaved
- ♠ Nivelul 5 : datele/informatia de paritate sunt imprastiate pe toate discurile
- ♠ Nivelul 6 : rezista la mai mult de o singura eroare de disc

Curs 12 (Administrarea informatiilor – fisiere) :

Structura (modul de interpretare a continutului) -> decis de program (UNIX) sau sistem de operare (Windows, OS/2). In al doilea caz, decizia se ia pe baza extensiei, i.e. sufixul din numele fisierului.

Tipuri de sisteme de fisiere :

- De uz general : NTFS (Windows) sau ext2fs / ext3fs / ext4fs, btrfs, zfs (Linux/UNIX)
- De uz particularizat : tmpfs (RAM disk), objfs (kernel debugging), procfs (interfata structuri procese)

Cluster = unitatea de masura pentru alocarea spatiului necesar memorarii unui fisier, la nivelul file-system-ului. -> Dimensiunea = o putere de forma 2^n sectoare.

Sisteme de fisiere jurnalizate -> NTFS si ext3fs/ext4f

Accesarea fisierelor in retele lan -> NFS

Sisteme de fisiere “next-generation” -> ZFS (Solaris), btrfs (Linux), ReFS (Windows), APFS (for Apple’s devices)