Gestiunea timpului, ceasul HW

In calculatorul nostru sunt 2 tipuri de ceasuri:

1. ceasul harwdare= componenta fizica in calculatorul nostru, un cip mai exact. Acest cip are memorie care stocheaza data (data in sens calendaristic). Partea interesanta la acest ceas harwdare este ca indiferent daca sistemul de calcul este pornit sau nu, ceasul functioneaza in continuare, fiind conectat la o baterie care nu are nevoie de alimentare(adica nu trebuie sa ai calculatorul bagat in priza). In cazul in care acea baterie este pe terminate, o sa transmita date eronate la fiecare boot.
2. ceasul sistem= este un software in kernel, care numara secundele de la 1 ianuarie 1970(unix epoch). Acest ceas este folosit in loc de ceasul hardware deoarece este muult mai rapid(ceasul sistem este incarcat in ram la boot). Diferenta dintre ceasul sistem si cel hardware este ca ceasul sistem se reseteaza de fiecare data la bootarea calculatorului. La bootare, ceasul sistem preia datele din memoria ceasului hardware si transforma acel timp in secunde(gen 1999273 secunde etc).

Cum stie calculatorul sa seteze ora EXACTA(localtime)?
In sistemele LInux la bootarea calculatorului ceasul hw transmite ora Londrei, iar dupa kernelul citeste din etc/localtime/ sa vada tara in care te afli si ajusteaza ceasul sistem in concordanta.
In sistemele Windows, ceasul hardware este fortat sa fie ajustat la ora tarii in care te afli.
In cazul in care ai si LInux si Windows apare un conflict deoarece windows are deja setata ora corecta in ceasul hardware iar linux stie ca trebuie sa faca +2 ore. Aceasta problema poate fi rezolvata totusi.

SISTEME DE FISIERE= o componenta a sistemului de operare, organizeaza fisierele

Sisteme de fisiere:-permanente(de tip hddd etc.)
-temporare
-de tip loop devices= O metodă inteligentă de a face un fișier obișnuit să se comporte ca un hard disk. (Exemplu: când deschizi un fișier `.iso`, sistemul îl vede ca pe un CD fizic, deși e doar un fișier pe disc)
-distribuite=fisiere stocate pe retea, nu local
-pseudo sisteme= nu stocheaza fisiere reale, ci ferestre catre kernel (de ex /sys care de fapt contine informatii live despre procese si hardware)

Când salvezi un fișier, sistemul trebuie să facă mai multe operații (pași) pe disc:

1. Să scrie datele propriu-zise.
2. Să actualizeze metadatele (nume, dimensiune, data).
3. Să marcheze spațiul ca fiind "ocupat" în lista de sectoare libere.

Scenariul de coșmar: Să zicem că pică curentul (sau dă crash kernelul) exact după pasul 1, dar înainte de pasul 2 și 3.

- Rezultat: Ai date scrise pe disc, dar sistemul nu știe că sunt acolo (inconsistență). Sau sistemul crede că ai un fișier, dar el arată către gunoi digital.

Soluția Veche (Fără Journaling): Trebuia să rulezi fsck (File System Check).
- Aceasta scana tot discul, sector cu sector, ca să compare ce e scris cu ce scrie în "cuprins".
- Dezavantaj: Pe discuri mari, dura ore întregi. Timp în care serverul era oprit (Downtime).

JOURNALING FS
Cum funcționează :
1. Zona Specială: Sistemul de fișiere își rezervă o zonă mică, circulară, numită Jurnal.
2. Tranzacția: Când vrei să modifici ceva, sistemul nu scrie direct în locul final de pe disc. Mai întâi, scrie o notă rapidă în Jurnal: "Intenționez să scriu fișierul X la adresa Y".
    - Aceasta se numește Tranzacție.
3. Commit (Comiterea): Imediat ce această notă este scrisă complet în Jurnal, tranzacția este considerată sigură ("committed").
4. Checkpointing (Scrierea reală): Într-un moment ulterior (când discul e mai liber), sistemul copiază liniștit datele din Jurnal la locul lor final pe disc.
5. Curățarea: După ce datele au ajuns cu bine la destinație, nota din Jurnal este ștearsă pentru a face loc altora.

avantaje:
Consistența Datelor (Fără erori):
- Dacă pică curentul înainte să scrie în Jurnal: E ca și cum operația nu a existat. Nu ai pierdut date vechi, nu ai corupt nimic.
- Dacă pică curentul după ce a scris în Jurnal, dar înainte să ajungă pe disc: Datele sunt salvate în Jurnal.

Recuperare Rapidă (Fast Recovery):
- La repornire, sistemul nu mai scanează tot discul (ore întregi).
- Se uită doar în Jurnal (care e mic).
- Dacă găsește tranzacții neterminate ("Replay Journal"), le execută rapid.
- Timpul de boot scade de la ore la secunde.

Temporary file systems (tmpfs)=functioneaza pe principiul: am nevoie sa rulez un program foarte mare si am nevoie neaparat de viteza asa ca il “plantez” in RAM intr-un mod organizat care emuleaza fisiere si directoare.

Deci, ca sa clarific, in memoria RAM sunt aruncate programele active, imaginile, tot ce ruleaza pe calculator, RAM-ul brut nu stie ce sunt alea fisiere sau directoare, ci doar adrese de memorie. TMPFS ii spune ram-ului hey ingradeste o zona din tine astfel incat sa se comporte ca un hard disk organizat in fisiere si directoare, deoarece am nevoie de viteza foarte mare pentru programul meu.

DE CE totusi?
Folosirea RAM-ului normal: Când scrii text, literele stau în RAM. Dar ele stau acolo ca "variabile" ale programului. Tu, ca utilizator, nu le poți vedea din afara programului. Nu poți deschide un "folder de RAM" să vezi literele alea.

Folosirea tmpfs: Dacă vrei să salvezi textul într-un fișier vizibil, dar vrei viteza RAM-ului, salvezi în /mnt/ramfs (care e tmpfs). Acum, acel text este tot în cipurile de memorie RAM, dar este împachetat ca un fișier. Sistemul de operare îl vede ca pe un fișier, îl poți copia, îi poți da drepturi de acces etc.

SISTEME DE FISIERE DISTRIBUITE
cum accesezi fișiere care nu se află fizic pe calculatorul tău, dar sistemul te păcălește să crezi că sunt acolo.
Acesta este conceptul de Sistem de Fișiere Distribuit (Distributed File System), iar exemplul clasic este NFS (Network File System).

Spre deosebire de discurile locale unde totul e "în casă", aici avem două roluri distincte care comunică prin mesaje (message passing):

- FS Server (Furnizorul):
    - Este calculatorul care are discul fizic.
    - El "exportă" (share-uiește) un director către rețea.
      de ex: serverul facultății care are temele studenților pe HDD-ul lui.
- FS Client (Consumatorul):
    - Este calculatorul tău.
    - El "importă" (montează) acel director.
    - Folosește VFS (Virtual File System) ca intermediar, astfel încât aplicațiile tale (editorul de text, browserul) să nu știe că fișierele vin prin cablul de rețea. Ele cred că sunt locale.

Tehnologia din spate: RPC (Remote Procedure Call)

Cum facem ca o comandă dată pe Client să se execute pe Server? Aici intervine RPC.

Conceptul de "Stub" (Dublura/Intermediarul): Să zicem că vrei să citești un fișier (read).

1. Pe Client: Tu apelezi funcția read(). Dar kernelul nu citește discul tău. El apelează un "Client Stub". Acesta este un "poștaș" care ia cererea ta, o împachetează și o trimite prin rețea.
2. Pe Server: Mesajul ajunge la "Server Stub". Acesta despachetează cererea și execută funcția read() reală pe discul serverului.
3. Retur: Rezultatul (conținutul fișierului) se întoarce pe același drum.

Marshalling / Unmarshalling (Traducerea datelor)

Calculatoarele pot fi diferite (unele citesc datele de la stânga la dreapta, altele invers - Big Endian vs Little Endian, sau au arhitecturi pe 32 vs 64 biți).

- Dacă trimiți numărul "5" brut prin rețea, un alt calculator l-ar putea înțelege ca "5000".
- Marshalling (Împachetarea): Transformarea parametrilor din formatul intern al Clientului într-un format standard de rețea (numit XDR - External Data Representation).
- Unmarshalling (Despachetarea): Serverul primește formatul XDR și îl traduce în formatul său intern pentru a-l putea procesa.