Sisteme de operare

*Aceasta **fituica** rezuma **Sisteme de operare**.*
In acest PDF poti vizualiza cuprinsul si bibliografia (daca sunt disponibile) si aproximativ doua pagini din documentul original.
*Arhiva completa de pe site contine un fisier, intr-un numar total de **74 pagini**.*
Fisierele documentului original au urmatoarele extensii: docx.

Extras

Subiectul 7: (exemplu de raspuns) Folosirea ASLR impiedica rularea de shellcode-uri de pe stiva prin pozitionarea stivei in diverse zone din spatiul de adresa; atacatorul nu poate sti (usor) care este adresa de start a shellcode-ului. Are eficienta in special pe sistemele pe 64 de biti. Pe cele pe 32 de biti se poate folosi brute force.

Subiectul 8: (exemplu de raspuns) Folosirea unui numar mare de thread-uri in cadrul unui proces poate conduce mai rapid la stack overflow. Fiecare thread are stiva proprie, cu dimensiunea fixata la crearea. Daca se creeaza prea multe thread-uri, se va ocupa foarte mult stiva. Stivele thread-urilor vor fi apropiate unele de altele astfel ca, in cazul unui flux de apeluri mare (apeluri recursive, de exemplu), exista riscul ca stiva unui thread sa suprascrie stiva altui thread.

Subiectul 11: (exemplu de raspuns) Blocurile/modulele implicate intro diagrama, despre care ar trebui discutat, sunt:

1.TCP communication

2.I/O model (async vs threading)

3.XML parser

Daca le luam de la coada la cap, parserul XML trebuie sa fie lightweight si sa implementeze un subset necesar comunicarii intre client si server.

Modelul de I/O, daca ne uitm la numarul maxim de clienti cerut de specificatii, de 1000, putem sa implementam folosind 1000 de threaduri care asculta pe acelasi socket. Daca insa ne uitam la faptul ca trebuie sa serveasca un numar relativ mare de tranzactii pe secunda, probabil ca un model async I/O ar fi mai potrivit, eliminand context switchurile dintre cele 1000 de threaduri din celalalt model. In cele din urma, probabil ca un model hibrind cu un thread care asculta pe un socket si foloseste async IO si un numar de threaduri care este o functie de numarul de core-uri este cel mai eficient.

Modulul de TCP are o particularitate interesanta. Din starea de listen, la aparitia unei conexiuni noi se creeaza un fd de date. TCPul fiind un protocol de tip stream, sunt doua posibilitati. Daca clientii trimit cereri care au toate nevoie de raspuns, atunci este ok, serverul citeste de pe socket pana cand se poate forma un mesaj XML corect, apoi il trimite mai departe catre procesare si trimite raspunsul inapoi pe aceeasi conexiune de date, dupa care inchid conexiunea. Cazul mai complicat este cand un client poate trimite notificari catre server fara sa astepte raspuns, caz in care este necesar sa se detecteze si sa se delimiteze mai multe requesturi in acelasi buffer. In acest caz este posibil sa apara si desincronizari intre client si server, daca clientul trimite notificari foarte rapid, bufferul de TCP poate sa contina requesturi incomplete, ceea ce complica detectia de mesaje bine formate.

Subiectul 12: (exemplu de raspuns) Aici problema consta in faptul ca nu este scalabil sa stochezi 1,000,000 de fisiere in acelasi director, si nici intro baza de date nu prea are sens sa stochezi ca bloburi toata povestea asta care insumeaza 10TB de date. Asadar, e clar ca fisierele vor trebui stocate pe disc in

foldere separate, cel mai bine intro structura arborescenta, care sa se poata intinde pe oricate filesystemuri, deoarece e vorba de o capacitate mai mare decat discurile uzuale. Asadar, blocurile implicate in aceasta solutie sunt doua:

1.Bloc de identificare

2.Bloc de stocare

Blocul de identificare se poate implementa cu o tabela simpla in orice sistem de baze de date, care face o mapare de la un identificator unic la o cale de fisier pe disc.

Blocul de stocare are un API simplu, prin care i se cere, pentru un nou fisier de stocat, locul in care se va stoca. Pentru a face asta, trebuie sa stie ce discuri exista in sistem, ce capacitati au si ce filesystem au, pentru a determina un numar optim de intrari in director pentru fiecare dintre ele. Cand se cere un slot pt un fisier nou, sistemul cauta pe volumele existente cel mai bun loc in termeni de spatiu disponibil, intrari in director, etc. Se poate implementa si un load balancer care sa distribuie fisierele cele mai cerute pe discuri diferite, etc.

14 iunie 2013

Subiectul 7: (exemplu de raspuns) Nu este recomandat sa se foloseasca mutex-uri intr-un handler de semnal. Daca se face lock pe mutex si, in programul principal, s-a facut de asemenea, lock pe mutex, exista riscul unui deadlock; handler-u de semnal intrerupe programul principal in timp ce acesta inca are ocupat mutex-ul.

Subiectul 8: (exemplu de raspuns) Un fisier mapat in memorie ocupa pagini fizice (frame-uri) care sunt mapate apoi in spatiuil de adresa al procesului. Un fisier poate fi mapat de mai multe procese, caz in care paginile fizice (frame-urile) aferente pot fi partajate. Scrierile in spatiul de adresa vor ajunge in spatiul fizic si vor fi flushed doar la apeluri specifice sau la inchiderea maparii.

Subiectul 11: (exemplu de raspuns) Pentru deschiderea unui fisier e nevoie sa localizam fisierul in dentry - costul este liniar in nr. de fisiere in director in ext2. Idee de baza: ca sa reducem timpul de acces, trebuie sa ne asiguram ca subiectele sunt stocate in directoare care au un numar redus de fisiere.

Pentru a implementa baza de date, avem nevoie de doua lucruri:

- o tabela de hash care mapeaza ID-ul fisierului in directorul care il contine. Aceasta tabela va fi stocata in memorie. Presupunand ca numarul de directoare este relativ mic, dimensiunea tabelei este data de nr de fisiere * (dim_id + pointer_nume_director) ~ 8 sau 16 octeti. 10 milioane de fisiere ar ocupa numai 160MB de RAM.

- o ierarhie de directoare in care fiecare director are un numar maxim de intrari X (prestabilit).

Se pot folosi multiple structuri ierarhice cu adancime prestabilita. Se creaza astfel mai multe directoare:

- un director "direct" care contine un numar de fisiere X

- un director "indirect" care contine X alte directoare fiecare cu X fisiere

- un director dublu indirect

- un director triplu indirect etc.

Se vor folosi directoarele "directe" dupa care cele indirecte, dublu-indirecte, etc.

Cum alegem X? Timpul de acces la un fisier depinde de X si de adancimea structurii de directoare, citirea unui director dureaza: X(logXN + 1). Stiind N (e.g. 10 milioane), se poate optimiza X alegand valoarea care

minimizeaza formula de mai sus.

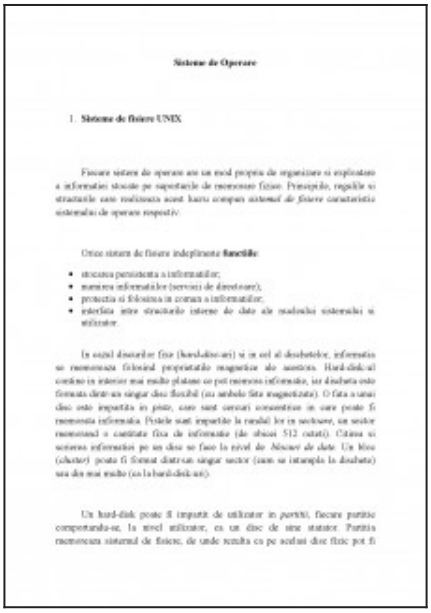
Subiectul 12: (exemplu de raspuns) Ideea principala este de a trata o cerere de continut cat mai repede si de a inchide conexiunea - astfel numarul de clienti conectati simultan este mic si acest lucru reduce stresul asupra resurselor sistemtului (thread-uri, descriptori).

Se poate folosi un pool de thread-uri; atunci cand vine o cerere noua se aloca cererea unuia din thread-urile din pool. Folosirea pool-ului de thread-uri minimizeaza costurile de startup, si reduc costurile de switching (no tlb flush), insa toti clientii vor folosi acelasi spatiu de adresa - totusi potentialele probleme de securitate par minore (read-only video data).

Se va folosi blocking API pt. citire din sockets - e cea mai usor de folosit. Non-blocking nu prea are sens cu thread-uri. Event based la fel. Cel mai probabil reteaua va deveni un bottleneck. Din cauza ca folosim un pool de thread-uri nr de thread-uri nu e o problema, si nici cel de descriptori (presupunand ca nu se executa accept atunci cand nu se poate repartiza cererea clientului unui thread).

. . . . . .
. . . . . .
. . . . . .

Imagini din documentul complet:



Sisteme de Operare

1. **Sisteme de fisiere UNIX**

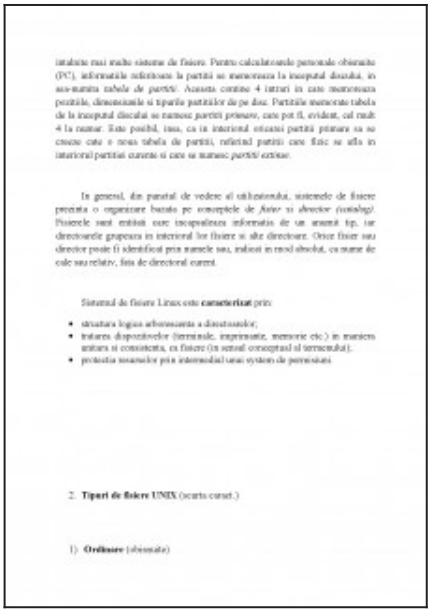
Fiecare sistem de operare are un mod propriu de organizare si exploatare a informatiei stocate pe suporturile de memorare fizice. Principiile, regulile si structurile care realizeaza acest lucru compun *sistemul de fisiere* caracteristic sistemului de operare respectiv.

Orice sistem de fisiere indeplineste **functiile**:

- stocarea persistenta a informatiilor;
- numirea informatiilor (servicii de directoare);
- protectia si folosirea in comun a informatiilor;
- interfata intre structurile interne de date ale nucleului sistemului si utilizator.

In cazul discurilor fixe (hard-disc-uri) si in cel al dischetelor, informatia se memoreaza folosind proprietatile magnetice ale acestora. Hard-disc-ul contine in interior mai multe platane pe care memoreaza informatia, iar dischetа este formata dintr-un singur disc flexibil (cu stratului fin magnetizat). O fata a unui disc este impartita in *piste*, care sunt cercuri concentrice in care poate fi memorata informatia. Pistele sunt impartite la randul lor in *sectoare*, un sector memorand o cantitate fixa de informatie (de obicei 512 octeti). Citirea si scrierea informatiei pe un disc se face la nivel de *blocuri de date*. Un bloc (cluster) poate fi format dintr-un singur sector (sunt sa intampla la dischetе) sau din mai multe (ca la hard-disk-uri).

Un hard-disk poate fi impartit de utilizator in *partitii*, fiecare partitie comportandu-se, la nivel utilizator, ca un disc de sine statator. Partitia memoreaza sistemul de fisiere, de unde rezulta ca pe acelasi disc fizic pot fi



intalnite mai multe sisteme de fisiere. Pentru calculatoarele personale obisnuite (PC), informatiile referitoare la partitii se memoreaza la inceputul discului, in asa-numita *tabela de partitii*. Aceasta contine 4 intrari in care memoreaza pozitiile, dimensiunile si tipurile partitiilor de pe disc. Partitiile memorate tabela de la inceputul discului se numesc *partitii primare*, care pot fi, evident, cel mult 4 la numar. Este posibil, insa, ca in interiorul oricarei partitii primare sa se creeze cate o noua tabela de partitii, referind partitii care fizic se afla in interiorul partitiei curente si care se numesc *partitii extinse*.

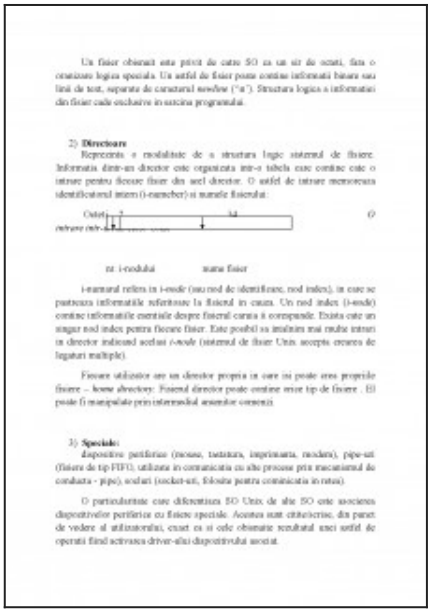
In general, din punctul de vedere al utilizatorului, sistemele de fisiere prezinta o organizare bazata pe conceptele de *fisier* si *director (catalog)*. Fisierele sunt entitati care incapsuleaza informatia de un anumit tip, iar directoarele grupeaza in interiorul lor fisiere si alte directoare. Orice fisier sau director poate fi identificat prin numele sau, indicand in mod absolut, ca nume de cale sau relativ, fata de directorul curent.

Sistemul de fisiere Linux este **caracterizat** prin:

- structura logica arborescenta a directoarelor;
- tratarea dispozitivelor (terminale, imprimante, memorie etc.) in maniera unitara si consistenta, ca fisiere (in sensul conceptual si terminologic);
- protectia resurselor prin intermediul unui system de permisiuni.

2. **Tipuri de fisiere UNIX** (scurta cursul.)

1) **Ordinare** (obisnuite)



Un fisier obisnuit este privit de catre SO ca un sir de octeti, fara o organizare logica speciala. Un astfel de fisier poate contine informatii binare sau linii de text, separate de caracterul newline ('\n'). Structura logica a informatiei din fisier cade exclusiv in sarcina programului.

2) **Directoare**

Reprezinta o modalitate de a structura logic sistemul de fisiere. Informatia dintr-un director este organizata intr-o tabela care contine cate o intrare pentru fiecare fisier din acel director. O astfel de intrare memoreaza identificatorul intern (i-number) si numele fisierului:

nr. i-nodului nume fisier

i-numarul refers to i-node (sau nod de identificare, nod index), in care se pastreaza informatiile referitoare la fisierul in cauza. Un nod index (i-node) contine informatii esentiale despre fisierul caruia ii corespunde. Exista cate un singur nod index pentru fiecare fisier. Este posibil sa intalnim mai multe intrari in director indicand acelasi *i-node* (sistemul de fisier Unix accepta crearea de legaturi multiple).

Fiecare utilizator are un director propriu in care isi poate crea propriile fisiere – *home directory*. Fiecare director poate contine orice tip de fisiere . El poate fi manipulate prin intermediul anumitor comenzi.

3) **Speciale:**

dispozitive periferice (tocuse, tastatura, imprimanta, modem), pipe-uri (fisiere de tip FIFO, utilizate in comunicatia cu alte procese prin mecanismul de conducta - pipe), socluri (socket-uri, folosite pentru comunicatia in retea).

O particularitate care diferentiaza SO Unix de alte SO este asocierea dispozitivelor periferice cu fisiere speciale. Acestea sunt caracterizate, din punct de vedere al utilizatorului, exact ca si cele obisnuite rezultatul unei astfel de operatii fiind activarea driver-ului dispozitivului asociat.

Mai multe detalii se gasesc in [pagina documentului](#) din Biblioteca.RegieLive.ro