\*\*Capitolul 1 - Introducere\*\*

- Rolul și funcțiile unui sistem de operare

- Structura și componentele unui sistem de operare

- Interfața utilizatorului și interfața programului

\*\*Capitolul 2 - Structuri ale sistemului de operare\*\*

- Monolitic vs. Stratificat vs. Microkernel

- Virtualizarea și mașini virtuale

- Modularitate și flexibilitate

\*\*Capitolul 3 - Procese\*\*

- Conceptul de proces

- Ciclul de viață al unui proces

- Comunicare și sincronizare între procese

- Procese întreruptibile vs. neîntreruptibile

\*\*Capitolul 4 - Memoria\*\*

- Tipuri de memorie: Primară vs. Secundară

- Organizarea memoriei fizice și logice

- Gestiunea spațiului de adresă

- Swapping și paginare

\*\*Capitolul 5 - Sincronizarea proceselor\*\*

- Curse de execuție (Race Conditions)

- Secțiuni critice și algoritmi de rezolvare

- Semnale și metode de sincronizare

- Probleme de sincronizare: Problema bufferelor limitate, Problema cititorilor-scriitorilor

\*\*Capitolul 6 - Planificarea CPU\*\*

- Algoritmi de planificare: FCFS, SJF, Round Robin, Priority

- Estimarea lungimii următoarei explozii de CPU

- Planificarea multiprocesor și metodele de împărțire a sarcinilor

\*\*Capitolul 7 - Deadlock-uri\*\*

- Condiții pentru apariția deadlock-ului

- Metode de prevenire și gestionare a deadlock-urilor

- Problemele de sincronizare: Problema bufferelor limitate, Problema cititorilor-scriitorilor

\*\*Capitolul 8 - Memorie Principală\*\*

- Registrul și limita, adresarea relativă

- Unitatea de gestionare a memoriei (MMU)

- Probleme de fragmentare și metode de rezolvare

\*\*Capitolul 9 - Memorie Virtuală\*\*

- Greșelile de paginare și gestionarea lor

- Strategii de încărcare a paginilor

- Memorie virtuală pură și tehnica Copy-on-Write (COW)

\*\*Capitolul 10 - Sisteme de Stocare în Masă\*\*

- Caracteristicile discurilor magnetice

- Tipuri de stocare atașate gazdei: EIDE, ATA, SATA, USB

- Algoritmi de planificare pentru discuri: FCFS, SSTF, SCAN

- Redundanța în stocarea RAID

\*\*Capitolul 11 - Interfața Sistemului de Fișiere\*\*

- Tabele de fișiere deschise, acces la fișiere și drepturi

- Organizarea directoarelor și a sistemelor de fișiere

- Montarea sistemelor de fișiere și drepturile de acces

\*\*Capitolul 12 - Implementarea Sistemului de Fișiere\*\*

- Alocarea blocurilor și gestionarea spațiului liber

- Tehnici de formatare și strategii de stocare a datelor

\*\*Capitolul 13 - Sisteme de Intrare/Ieșire\*\*

- Rolul și componentele unui sistem de I/O

- Controlul dispozitivelor și driver-ele

- Metode de acces la dispozitive: I/O memorie-mapelată, I/O bazată pe întreruperi

- Diferențe între dispozitivele de bloc și cele de caractere

\*\*Capitolul 14 - Protecție\*\*

- Principiul privilegiilor minime

- Matricea de acces și conceptul de domeniu

- Protecție bazată pe limbaje și implementarea ei în sistemele de operare

\*\*Capitolul 15 - Securitate\*\*

- Metode de violare: Masquerading, Man-in-the-middle, Session hijacking

- Tehnici de securitate: Criptografie, Firewall, Autentificare, Certificate digitale, Detectarea intruziunilor

**Capitolul 1 – Introducere**

• Un sistem de operare (OS) este un program care acționează ca intermediar între un utilizator al unui computer și hardware-ul computerului.

• Obiective: Executarea programelor utilizatorului, facilitarea utilizării sistemului de calcul, utilizarea eficientă a hardware-ului.

• Sistem de calcul: Hardware ↔ OS ↔ Aplicații ↔ Utilizatori (↔ = 'folosește')

• OS este:

◦ Alocatar de resurse: decide între cererile conflictuale pentru utilizarea eficientă și echitabilă a resurselor

◦ Program de control: controlează execuția programelor pentru a preveni erori și utilizare improprie a computerului

• Nucleul (Kernel): singurul program care rulează în permanență pe computer.

• Programul de pornire (Bootstrap program): încărcat la pornire sau repornire

◦ Stocat în ROM sau EPROM (cunoscut sub numele de firmware), inițializează toate aspectele sistemului, încarcă nucleul OS și începe execuția

• I/O și CPU pot executa în mod concurent

• Controlerele de dispozitive informează CPU-ul că a terminat operația prin cauzarea unei întreruperi

◦ Întreruperea transferă controlul la rutina de service pentru întreruperi în general, prin vectorul de întreruperi, care conține adresele tuturor rutinelor de service

◦ Întreruperile intrante sunt dezactivate în timp ce o altă întrerupere este procesată

◦ Captura (Trap) este o întrerupere generată de software cauzată de eroare sau cerere a utilizatorului

◦ OS determină tipul de întrerupere care a avut loc prin interogare sau sistemul de întreruperi vectorizat

• Apel de sistem: solicitare către sistemul de operare pentru a permite utilizatorului să aștepte finalizarea I/O

• Tabelul de stare a dispozitivului: conține o intrare pentru fiecare dispozitiv I/O indicând tipul, adresa și starea acestuia

◦ OS indexează în tabelul dispozitivului I/O pentru a determina starea dispozitivului și pentru a modifica intrarea tabelului pentru a include o întrerupere

• Structura de stocare:

◦ Memorie principală - acces aleatoriu, volatilă

◦ Stocare secundară - extensie a memoriei principale care furnizează o stocare mare, non-volatilă

◦ Disk - împărțit în piste(tracks) care sunt subdivizate în sectoare. Controlerul de disc determină interacțiunea logică dintre dispozitiv și computer.

• Caching - copierea informațiilor într-un sistem de stocare mai rapid

• Sisteme multiprocesor: Creșterea throughput-ului, economie de scară, creșterea fiabilității

◦ Pot fi asimetrice sau simetrice

◦ Sisteme grupate - Sisteme multiprocesor legate

• Multiprogramare - Oferă eficiență prin programarea sarcinilor

◦ Când OS trebuie să aștepte (de exemplu, pentru I/O), trece la o altă sarcină

• Timesharing - CPU comută sarcinile atât de frecvent încât fiecare utilizator poate interacționa cu fiecare sarcină în timp ce rulează (computație interactivă)

• Funcționare duală permite OS-ului să se protejeze pe sine și alte componente ale sistemului - Modul utilizator și modul kernel

◦ Unele instrucțiuni pot fi executate doar în modul kernel, acestea sunt privilegiate

• Procesele cu un singur fir de execuție au un singur contor de program, iar procesele cu mai multe fire de execuție au un contor de program per fir

• Protecția - mecanism pentru controlul accesului proceselor sau utilizatorilor la resursele definite de OS

• Securitatea - apărarea unui sistem împotriva atacurilor

• ID-urile utilizatorului (UID), unul per utilizator, și ID-urile de grup determină care utilizatori și grupuri de utilizatori au care privilegii.

**Capitolul 2 - Structuri de Sistem de Operare**

• Interfața utilizatorului (UI) - Poate fi Linie de Comandă (CLI) sau Interfață Grafică a Utilizatorului (GUI) sau în lot (Batch).

◦ Acestea permit utilizatorului să interacționeze cu serviciile sistemului prin apeluri de sistem (de obicei scrise în C/C++).

• Alte servicii de sistem utile utilizatorului includ: execuția programelor, operațiile I/O, manipularea sistemului de fișiere, comunicările și detectarea erorilor.

• Serviciile care există pentru a asigura funcționarea eficientă a OS sunt: alocarea resurselor, contabilitatea, protecția și securitatea.

• Cele mai multe apeluri de sistem sunt accesate printr-o Interfață de Program a Aplicației (API) precum Win32, POSIX, Java.

• De obicei, fiecărui apel de sistem îi este asociat un număr.

◦ Interfața de apeluri de sistem menține o tabelă indexată conform acestor numere.

• Parametrii pot fi necesari să fie transmiși OS-ului în timpul unui apel de sistem și pot fi făcuți prin:

◦ Transmiterea în registre, adresa parametrului stocată într-un bloc, împinsă pe stivă de către program și scoasă de OS.

◦ Metodele bloc și stivă nu limitează numărul sau lungimea parametrilor transmiși.

• Apelurile de sistem de control al procesului includ: terminare, avortare, încărcare, executare, creare/terminare proces, așteptare, alocare/eliberare memorie.

• Apelurile de sistem de gestionare a fișierelor includ: creare/ștergere fișier, deschidere/închidere fișier, citire, scriere, obținere/setare atribute.

• Apelurile de sistem de gestionare a dispozitivelor includ: cerere/eliberare dispozitiv, citire, scriere, atașare/detașare logică a dispozitivelor.

• Apelurile de sistem de întreținere a informațiilor includ: obținere/setare oră, obținere/setare date sistem, obținere/setare atribute proces/fișier/dispozitiv.

• Apelurile de sistem de comunicații includ: creare/ștergere conexiune de comunicare, trimitere/recepție, transfer de stare.

• Abordare stratificată a sistemului de operare:

◦ Sistemul de operare este împărțit în mai multe straturi (nivele), fiecare construit pe niveluri inferioare. Cel mai de jos strat (stratul 0) este hardware-ul, cel mai înalt (stratul N) este interfața utilizatorului.

◦ Cu modularitatea, straturile sunt selectate astfel încât fiecare să utilizeze funcții (operații) și servicii numai din straturi de nivel inferior.

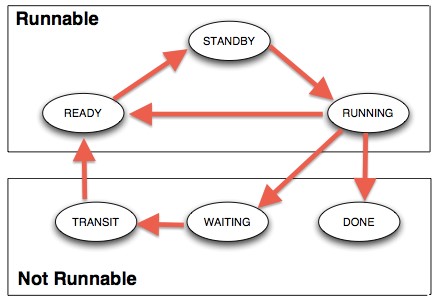
• Mașină virtuală: utilizează abordarea stratificată, tratează hardware-ul și nucleul OS ca și cum ar fi hardware-ul.

◦ Gazda creează iluzia că un proces are propriul său procesor și propria memorie virtuală.

◦ Fiecare oaspete este furnizat cu o copie 'virtuală' a computerului de bază.

• Defecțiunile aplicației pot genera un fișier core dump care capturează memoria procesului.

• Defecțiunile sistemului de operare pot genera un fișier crash dump care conține memoria kernelului.

**Capitolul 3 – Procese**

• Procesul conține un program counter, stack și secțiune de date.

◦ Secțiunea Text: codul programului în sine

◦ Stack: date temporare (parametrii funcției, adresele de retur, variabile locale)

◦ Secțiunea de date: variabile globale

◦ Heap: conține memoria alocată dinamic în timpul rulării

• Blocul de control al procesului (PCB): conține informații asociate cu fiecare proces: starea procesului, PC, registrele CPU, informații despre planificare, informații contabile, informații despre starea I/O

• Tipuri de procese:

◦ I/O Bound: petrece mai mult timp efectuând I/O decât calcule, multe rafale scurte de CPU

◦ CPU Bound: petrece mai mult timp efectuând calcule, câteva rafale foarte lungi de CPU

• Când CPU trece la alt proces, sistemul trebuie să salveze starea vechiului proces (în PCB) și să încarce starea salvată (din PCB) pentru noul proces printr-un context switch

◦ Timpul unui context switch depinde de hardware

• Procesele părinte creează procese copil (formează un arbore)

◦ PID permite gestionarea proceselor

◦ Părinții și copiii pot partaja toate/unele/niciunul dintre resurse

◦ Părinții pot executa în mod concurent cu copiii sau pot aștepta până când copiii se termină

◦ Apelul de sistem fork() creează un nou proces

▪ Apelul de sistem exec() este folosit după un fork pentru a înlocui spațiul de memorie al proceselor cu un nou program

• Procesele cooperante necesită comunicare între procese (IPC): memorie partajată sau transmitere de mesaje

• Transmiterea mesajelor poate fi blocantă sau non-blocantă

◦ Blocantă este considerată sincronă

▪ Transmiterea blocantă determină expeditorul să blocheze până când mesajul este primit

▪ Primirea blocantă determină receptorul să blocheze până când un mesaj este disponibil

◦ Non-blocantă este considerată asincronă

▪ Transmiterea non-blocantă determină expeditorul să trimită mesajul și să continue

▪ Primirea non-blocantă determină receptorul să primească un mesaj valid sau nul

Proces Zombie ~ Un proces care și-a terminat execuția, dar încă există în tabelul de procese.

Procesul zombie rămâne în acest stadiu pentru a permite părintelui să își verifice starea de terminare.

Odată ce părintele își finalizează verificările, sistemul de operare elimină complet procesul zombie.

Un proces devine zombie atunci când își finalizează execuția, dar părintele său nu a primit încă informații despre terminare.

Proces Orfan ~ Un proces care își pierde părintele înainte de a-și finaliza execuția.

De obicei, în sistemele de operare, un părinte așteaptă terminarea copiilor săi folosind apeluri de sistem specifice.

Dacă părintele unui proces moare sau se încheie înainte ca procesul său copil să își finalizeze execuția, acel copil devine un proces orfan.

În mod obișnuit, un proces orfan este preluat de către procesul "init" sau un alt proces "adopție" în sistem.

**Capitolul 4 - Fire de execuție (Threads)**

• Firele de execuție reprezintă unitatea fundamentală de utilizare a CPU care stă la baza sistemelor de calcul cu fire multiple

• Crearea proceselor este o operație grea, în timp ce crearea firelor de execuție este ușoară

◦ Poate simplifica codul și crește eficiența

• Kernelurile sunt în general multithreading

• Modelele de multithreading includ: Mulți-la-Unu, Unu-la-Unu, Mulți-la-Mulți

◦ Mulți-la-Unu: Mulți fire de execuție la nivel de utilizator mapate pe un singur fir de execuție la nivel de kernel

◦ Unu-la-Unu: Fiecare fir de execuție la nivel de utilizator se mapează pe un fir de execuție la nivel de kernel

◦ Mulți-la-Mulți: Mulți fire de execuție la nivel de utilizator mapate pe mulți fire de execuție la nivel de kernel

• Biblioteca de fire furnizează programatorului un API pentru crearea și gestionarea firelor de execuție

• Problemele includ: anularea firelor de execuție, gestionarea semnalelor (sincron/asincron), gestionarea datelor specifice firelor de execuție și activarea programelor planificatoare.

◦ Anularea:

▪ Anularea asincronă încheie imediat firul de execuție țintă

▪ Anularea amânată permite firului de execuție țintă să verifice periodic dacă trebuie anulat

◦ Procesorul de semnale procesează semnalele generate de un eveniment specific, livrate unui proces, gestionat

◦ Activările planificatorului oferă apeluri ascendente - un mecanism de comunicare de la kernel la biblioteca firelor de execuție.

▪ Permite aplicației să mențină numărul corect de fire de execuție ale kernelului

---

Semafoare:

• Semafoarele sunt obiecte de sincronizare care controlează accesul concurent la resurse

• Pot fi utilizate pentru a evita condițiile de cursă și a asigura sincronizarea între firele de execuție

Variabile conditionate:

• Variabilele conditionate oferă un mecanism pentru așteptarea sau semnalarea unui anumit eveniment între mai multe fire de execuție

• Pot fi utilizate pentru a coordona execuția firelor de execuție și pentru a evita blocarea inutilă

Nota: Semnul "/" (bară oblică) reprezintă un spațiu pentru a face textul mai ușor de citit

**Capitolul 5 - Sincronizarea Proceselor**

• Curse de Condiție (Race Condition): Mai multe procese accesează și manipulează aceleași date în mod concurent, rezultatul depinde de ordinea în care au loc aceste accesări.

• Fiecare proces are o secțiune critică de cod, unde manipulează datele.

- Pentru a rezolva problema secțiunii critice, fiecare proces trebuie să solicite permisiunea de a intra în secțiunea critică în secțiunea de intrare, să urmeze secțiunea critică cu secțiunea de ieșire și apoi să execute secțiunea de rest.

- Este dificil să se rezolve această problemă în kernel-uri preemptive.

• Soluția lui Peterson: soluție pentru două procese.

do {

flag[i] = TRUE;

turn = j;

while (flag[j] && turn == j)

/\*secțiunea critică\*/

flag[i] = FALSE;

/\*secțiunea de rest\*/

} while (TRUE);

- Cele două procese partajează două variabile: int turn și Boolean flag[2].

- turn: ale cui este rândul să intre în secțiunea critică.

- flag: indicație dacă un proces este pregătit să intre în secțiunea critică.

- flag[i] = true indică faptul că procesul Pi este pregătit.

- Algoritmul pentru procesul Pi:

• Mașinile moderne furnizează instrucțiuni hardware atomice: Atomice = non-interruptabile.

do {

dobândirea

/\*secțiunea critică\*/

eliberarea

/\*restul\*/

} while (TRUE);

• Soluție folosind Lucruri (Locks):

• Soluție folosind Test-And-Set: Variabilă booleană partajată "lock", inițializată la FALSE.

do {

while (TestAndSet(&lock))

// nu face nimic

/\* secțiunea critică\*/

lock = FALSE;

// secțiunea de rest

} while (TRUE);

boolean TestAndSet(boolean \*target) {

boolean rv = \*target;

\*target = TRUE;

return rv;

}

• Soluție folosind Swap: Variabilă booleană partajată "lock", inițializată la FALSE; Fiecare proces are variabila locală booleană "key".

do {

key = TRUE;

while (key == TRUE)

Swap(&lock, &key);

// secțiunea critică

lock = FALSE;

// secțiunea de rest

} while (TRUE);

void Swap(boolean \*a, boolean \*b) {

boolean temp = \*a;

\*a = \*b;

\*b = temp;

}

• Alte probleme de sincronizare includ Problema Bufferului Delimitat și Problema Scriitorilor-Cititori.

• Monitorul este o abstractizare la un nivel înalt care oferă un mecanism convenabil și eficient pentru sincronizarea proceselor.

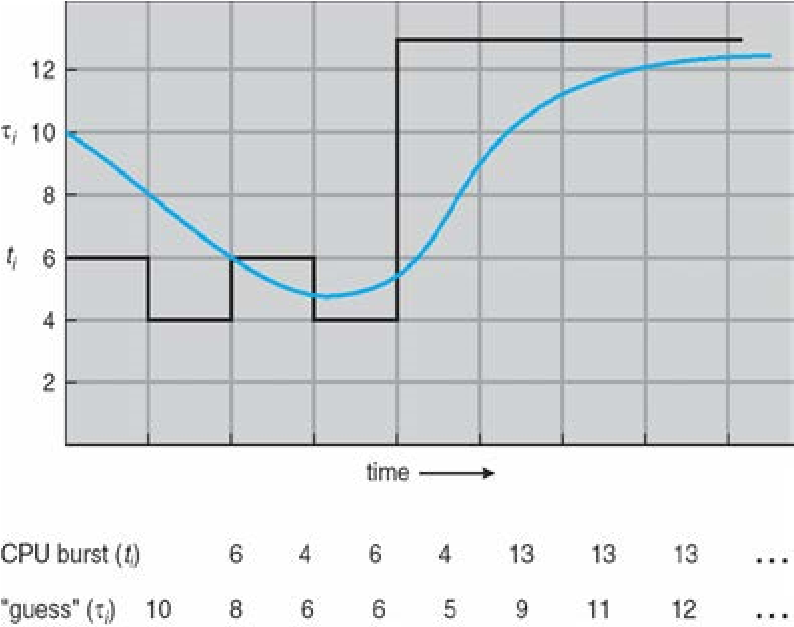
- Doar un singur proces poate fi activ în cadrul monitorului la un moment dat.

- Poate utiliza variabile de condiție pentru a suspenda și relua procesele (de exemplu, condițiile x, y).

- x.wait() - un proces care invocă operația este suspendat până la x.signal().

- x.signal() - reia unul dintre procesele (dacă există) care a invocat x.wait().

- Poate fi implementat cu semafoare.

**Capitolul 6 - Planificarea CPU-ului**

• Execuția procesului constă într-un ciclu de execuție CPU și așteptare I/O.

• Deciziile de planificare a CPU-ului au loc atunci când un proces:

- Trece de la rulare la așteptare (non-preemptiv)

- Trece de la rulare la gata (preemptiv)

- Trece de la așteptare la gata (preemptiv)

- Se termină (non-preemptiv)

• Modulul de dispacher oferă controlul CPU-ului procesului selectat de planificatorul pe termen scurt.

- Latența de dispecerizare - timpul necesar pentru dispacher pentru a opri un proces și a începe altul.

• Algoritmii de planificare sunt aleși pe baza criteriilor de optimizare (ex: throughput, turnaround time, etc.)

- FCFS, SJF, Shortest-Remaining-Time-First (preemptiv SJF), Round Robin, Priority.

• Determinarea lungimii următorului burst CPU: Medie Exponențială:

1. tn = lungimea reală a enulei burst CPU.

2. τn+1 = valoarea prognozată pentru următorul burst CPU.

3. α, 0 ≤ α ≤ 1 (în mod obișnuit, α este setat la 1/2).

4. Se definește: τn+1 = α\*tn + (1-α)τn.

• Planificarea pe bază de priorități poate duce la înfometare, care poate fi rezolvată prin

îmbătrânirea unui proces (pe măsură ce trece timpul, se crește prioritatea).

• În Round Robin, intervalele de timp mici pot duce la un număr mare de

comutări de context.

- Cuantumul de timp ar trebui ales astfel încât 80% din procese să aibă

timpuri de burst mai scurte decât cuantumul de timp.

• Cozi Multinivel și Cozi Multinivel cu Feedback au multiple

cozi de procese cu diferite niveluri de prioritate.

- În coada cu feedback, prioritatea nu este fixată → Procesele pot fi promovate sau retrogradate la cozi diferite.

- Cozile de feedback pot avea algoritmi de planificare diferiți la diferite niveluri.

• Planificarea Multiprocesor se realizează în mai multe moduri diferite:

- Multiprocesare asimetrică: doar un procesor accesează structurile de date ale sistemului → nu este nevoie de partajare a datelor.

- Multiprocesare simetrică: fiecare procesor este auto-planificator (în prezent, metoda cea mai comună).

- Afinitatea procesorului: un proces care rulează pe un procesor are mai multe șanse să continue să ruleze pe același procesor

(astfel încât memoria procesorului să conțină în continuare date specifice acelui proces).

• Formula lui Little poate ajuta la determinarea timpului mediu de așteptare per proces în orice algoritm de planificare:

- n = λ x W

- n = lungimea medie a cozii; W = timpul mediu de așteptare în coadă; λ = rata medie de sosire în coadă.

• Simulările sunt modele programate ale unui sistem de calcul cu ceasuri variabile.

- Folosite pentru a obține statistici care indică performanța algoritmului.

- Rularea simulărilor este mai precisă decât modelele de așteptare (ca Legea lui Little).

- Cu toate că este mai precisă, are costuri și riscuri ridicate.

**Capitolul 7 - Blocaje**

• Caracteristici ale Blocajelor: un blocaj poate apărea dacă aceste condiții sunt îndeplinite simultan

- Excludere Mutuă: doar un proces la un moment dat poate utiliza un resursă.

- Ținere și Așteptare: procesul care deține o resursă așteaptă să obțină resursa deținută de un alt proces.

- Nepreemțiune: o resursă poate fi eliberată doar de către procesul care o deține, după ce acesta și-a terminat sarcina.

- Așteptare Circulară: un set de procese aflate în așteptare astfel încât Pn-1 așteaptă resursa de la Pn, iar Pn așteaptă pe P0.

▪ "Filozofii la Cină" în blocaj

**Capitolul 8 - Memoria Principală**

• Cache se află între memoria principală și registrele CPU.

• Registrele de bază și limită definesc spațiul de adrese logice utilizabil de către un proces.

• Adresele codului compilat se leagă la adrese relocabile.

◦ Pot apărea la trei etape diferite

▪ La timpul compilării: dacă locația în memorie este cunoscută a priori, se poate genera cod absolut.

▪ La timpul încărcării: trebuie generat cod relocabil dacă locația în memorie nu este cunoscută la compilare.

▪ La timpul execuției: legarea este amânată până la rulare dacă procesul poate fi mutat în timpul execuției.

• Unitatea de management a memoriei (MMU) este un dispozitiv care face maparea între adresele virtuale și cele fizice.

• Un sistem simplu folosește un registru de relocare care adaugă doar o valoare de bază la adrese.

• Swapping permite ca spațiul total de memorie fizică al proceselor să depășească memoria fizică disponibilă.

◦ Def: procesul este temporar schimbat într-un magazie de suport și apoi readus pentru a continua execuția.

• Magazia de suport: disc rapid suficient de mare pentru a găzdui copii ale tuturor imaginilor de memorie.

• Roll out, roll in: variantă de schimbare pentru programarea bazată pe prioritate.

◦ Procesul cu prioritate mai mică este schimbat pentru a încărca procesul cu prioritate mai mare.

• Soluții la Problema de Alocare Dinamică a Stocării:

◦ Primul venit: alocă prima adâncitură suficient de mare.

◦ Cel mai potrivit: alocă cea mai mică adâncitură suficient de mare (trebuie căutată întreaga listă) → cea mai mică adâncitură rămasă.

◦ Cel mai nefavorabil: alocă cea mai mare adâncitură (căutare întreaga listă) → cea mai mare adâncitură rămasă.

• Fragmentare externă: spațiul total de memorie există pentru a satisface cererea, dar nu este contiguu.

◦ Redusă prin compactare: relocarea memoriei libere pentru a fi grupate într-un singur bloc.

▪ Posibilă numai dacă relocarea este dinamică.

• Fragmentare internă: memoria alocată poate fi puțin mai mare decât memoria cerută.

• Memoria fizică este împărțită în cadre de dimensiuni fixe: dimensiunea este o putere a lui 2, între 512 octeți și 16 MB.

• Memoria logică este împărțită în blocuri de aceeași dimensiune: pagini.

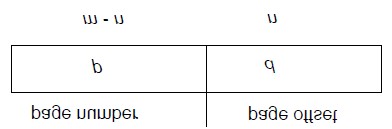
• Tabela de pagini este folosită pentru a traduce adresele logice în adrese fizice.

◦ Numărul paginii (p): folosit ca index într-o tabelă de pagini.

◦ Offsetul paginii (d): combinat cu adresa de bază pentru a defini adresa fizică a memoriei.

• Lista cadrelor libere este menținută pentru a ține evidența cadrului care pot fi alocate.

Pentru o anumită adresă logică de spațiu 2m și dimensiunea paginii 2n.



• Buffer-ul de căutare rapidă în tranzit (TLB) este un cache al CPU-ului pe care hardware-ul de gestionare a memoriei îl folosește pentru a îmbunătăți viteza de traducere a adreselor virtuale.

◦ De obicei, mic - între 64 și 1024 de intrări.

◦ La pierderea în TLB, valoarea este încărcată pentru un acces mai rapid data viitoare.

◦ TLB este asociativ - căutat în paralel.

• Timpul efectiv de acces: EAT = (1 + ε) α + (2 + ε)(1 – α)

◦ ε = unitatea de timp, α = rata de lovituri.

• Biții de validitate și invaliditate pot fi utilizați pentru a proteja memoria.

◦ "Valid" dacă pagina asociată este în spațiul de adrese logice al procesului, astfel este o pagină legală.

• Se pot avea tabele de pagini cu mai multe niveluri (tabele de pagini paginabile).

• Tabelele de pagini hash: numărul virtual al paginii este dispersat în tabela de pagini.

◦ Tabela de pagini are un lanț de elemente care se dispersează în aceeași locație.

◦ Fiecare element are (1) numărul virtual al paginii, (2) valoarea cadrului de pagină mapat, (3) un pointer către elementul următor.

◦ Se caută în lanț pentru numărul virtual al paginii.

• Tabela de segment - mapază adresele fizice bidimensionale.

◦ Intrările sunt protejate cu biți de validitate și privilegii r/w/x.

**Capitolul 9 - Memorie Virtuală**

• Memorie virtuală: separarea memoriei logice utilizatorului de memoria fizică.

◦ Doar o parte a programului trebuie să fie în memorie pentru execuție → spațiul de adrese logic > spațiul de adrese fizic.

◦ Permite spațiilor de adrese să fie partajate de mai multe procese → mai puțin swapping.

◦ Permite paginilor să fie partajate în timpul fork(), accelerând crearea de procese.

• Page fault apare prima dată când există o referință la o pagină specifică → declanșează OS-ul.

◦ Trebuie să decidă să abandoneze dacă referința este invalidă sau dacă pagina dorită nu este încă în memorie

▪ Dacă este cea din urmă: obține un cadru gol, schimbă pagina în cadru, resetează tabelele pentru a indica că pagina este acum în memorie, setează bitul de validare, reia instrucția care a cauzat page fault.

◦ Dacă o instrucțiune accesează mai multe pagini apropiate → mai puțin "durere" datorită localității de referință.

• Demand Paging aduce o pagină în memorie doar atunci când este necesar → mai puțin I/O și memorie necesară.

◦ Swapper leneș - nu schimbă niciodată o pagină în memorie decât dacă pagina va fi necesară.

◦ Ar putea duce la un număr mare de page-faulturi.

◦ Performanță: EAT = [(1-p)\* acces la memorie + p\*(suprasarcina page fault + swap page out + swap page in + repornire suprasarcină)]; unde rata de page fault este 0 ″ p ″ 1

▪ dacă p = 0, niciun page fault; dacă p = 1, fiecare referință este o eroare.

◦ Se poate optimiza demand paging prin încărcarea întregii imagini a procesului în spațiul de schimb în momentul încărcării procesului

• Pure Demand Paging: procesul începe fără pagini în memorie.

• Copiere la scriere (Copy-on-Write - COW) permite proceselor părinte și copil să partajeze inițial aceleași pagini în memorie.

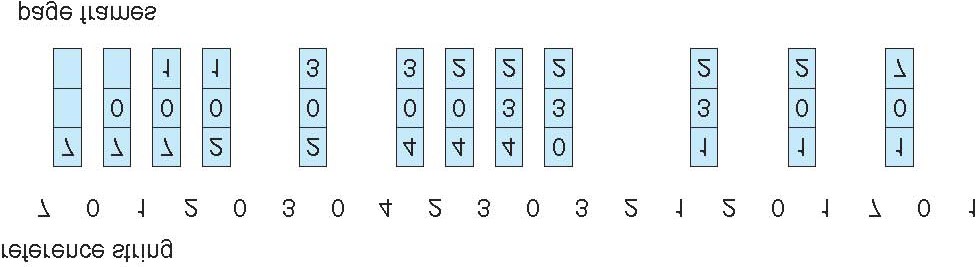
◦ Dacă unul dintre procese modifică o pagină partajată, pagina este copiată doar atunci.

• Bitul de modificare (dirty) poate fi utilizat pentru a reduce supraîncărcarea transferurilor de pagini → doar paginile modificate sunt scrise pe disc.

• Când o pagină este înlocuită, se scrie pe disc dacă a fost marcată ca murdară și se înlocuiește cu pagina dorită.

• Paginile pot fi înlocuite folosind diferite algoritme: FIFO, LRU (mai jos)

◦ Stiva poate fi folosită pentru a înregistra cele mai recente referințe la pagini (LRU este un algoritm "stack"



◦ Algoritmul de șansă secundară folosește un bit de referință.

▪ Dacă este 1, se decrementează și rămâne în memorie.

▪ Dacă este 0, se înlocuiește cu pagina următoare.

• Alocare fixă a paginilor: Alocare proporțională – Alocă în funcție de dimensiunea procesului

◦ si = dimensiunea procesului Pi, S = Σsi, m = numărul total de cadre, ai – alocare pentru Pi

◦ ai = (si/S)\*m

• Înlocuire globală: procesul selectează un cadru de înlocuire din totalitatea cadrelor.

◦ Un proces poate lua un cadru de la altul.

◦ Timpul de execuție al procesului poate varia în mod semnificativ.

◦ O mai mare capacitate de procesare.

• Înlocuire locală: fiecare proces selectează doar din propria sa mulțime de cadre alocate.

◦ Performanță mai constantă.

◦ Posibilă sub-utilizare a memoriei.

• Rata de page fault este foarte mare dacă un proces nu are "suficiente" pagini.

◦ Trashing: un proces este ocupat cu schimbul de pagini → se efectuează minim de muncă efectivă.

• I/O Interlock: Paginile trebuie uneori blocate în memorie.

**Capitolul 10 - Sisteme de Stocare de Masă**

• Discurile magnetice asigură cea mai mare parte a stocării secundare - se rotesc între 60 și 250 de ori pe secundă.

◦ Rata de transfer: rata la care datele curg între unitatea de stocare și computer.

◦ Timpul de poziționare (timpul de acces aleatoriu) este timpul necesar pentru a muta brațul dispozitivului de disc către cilindrul dorit (timpul de căutare) și timpul necesar pentru

ca sectorul dorit să rotească sub capul dispozitivului de disc (latența de rotație).

◦ Head crash: capul dispozitivului de disc face contact cu suprafața discului.

• Unitatea este conectată la bus-ul I/O al computerului - EIDE, ATA, SATA, USB, etc.

◦ Controllerul gazdă folosește bus-ul pentru a comunica cu controllerul dispozitivului de disc.

• Latența de acces = Timpul mediu de acces = timpul mediu de căutare + latența medie (rapid ~5ms, lent ~14.5ms).

• Timpul mediu de I/O = timpul mediu de acces + (cantitatea de transfer / rata de transfer) + suprancărcarea controllerului.

◦ Exemplu: pentru a transfera un bloc de 4KB pe un disc de 7200 RPM cu un timp mediu de căutare de 5ms, o rată de transfer de 1Gb/sec cu o suprancărcare a controllerului de 0.1ms = 5ms + 4.17ms + 4KB / 1Gb/sec + 0.1ms = 9.27ms + 0.12ms = 9.39ms.

• Unitățile de disc sunt adresate ca matrice unidimensională de blocuri logice.

◦ Matricea unidimensională este mapată în sectoarele discului în ordine secvențială.

• Stocarea conectată la gazdă este accesată prin porturi I/O care comunică cu bus-urile I/O.

◦ Rețea de stocare (SAN): multe gazde se conectează la multe unități de stocare, comun în medii de stocare mari.

- ▪ Stocarea este făcută disponibilă prin mascarea LUN de la aranjamente specifice la servere specifice.

• Stocarea conectată la rețea (NAS): stocarea este făcută disponibilă printr-o rețea în loc de o conexiune locală.

• În programarea planificării discurilor, dorim să minimizăm timpul de căutare; Timpul de căutare este proporțional cu distanța de căutare.

• Lățimea de bandă este (numărul total de octeți transferați) / (timpul total între prima cerere și finalizarea ultimului transfer).

• Sursele de cereri de I/O pentru disc: OS-ul, procesele sistemului, procesele utilizatorului.

◦ OS-ul menține o coadă de cereri, pe disc sau pe dispozitiv.

• Există mai multe algoritme pentru a planifica servirea cererilor de I/O pentru disc.

◦ FCFS, SSTF (cel mai scurt timp de căutare primul), SCAN, CSCAN, LOOK, CLOOK

▪ SCAN/elevator: brațul începe la un capăt și se mișcă spre celălalt, servind

cererile pe măsură ce avansează, apoi revine.

▪ CSCAN: în loc să revină, merge imediat înapoi la început.

▪ LOOK/CLOOK: Brațul merge doar până la ultima cerere în fiecare direcție, apoi

revine imediat.

• Formatarea la nivel fizic/înalt: împărțirea unui disc în sectoare pe care controllerul dispozitivului de disc le poate citi și scrie - de obicei, 512 octeți de date

• Partiționare: împărțirea discului în una sau mai multe grupuri de cilindri, fiecare tratat ca un disc logic.

• Formatare logică: "crearea unui sistem de fișiere".

• Creșterea eficienței prin gruparea blocurilor în clustere - I/O pe disc este efectuată pe blocuri.

◦ Blocul de pornire inițializează sistemul - încărcătorul de pornire este stocat în blocul de pornire.

• Spațiul de swap: memoria virtuală folosește spațiul de disc ca o extensie a memoriei principale.

◦ Kernelul folosește hărți de swap pentru a urmări utilizarea spațiului de swap.

• RAID: Mai multe unități de discuri oferă fiabilitate prin redundanță - crește timpul mediu până la eșec.

◦ Disk striping utilizează grup de discuri ca o unitate de stocare.

◦ Mirroring/shadowing (RAID 1) - păstrează duplicat al fiecărui disc.

◦ Striping cu oglinzi (RAID 1+0) sau oglinzi striate (RAID 0+1) oferă performanță/fiabilitate ridicată.

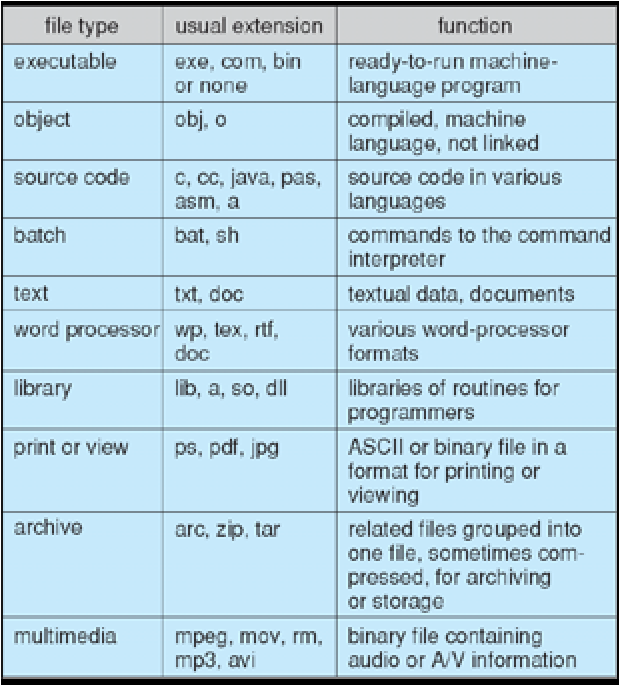
◦ Paritatea între blocuri (RAID 4, 5, 6) folosește mult mai puțină redundanță.

• Solaris ZFS adaugă sume de control pentru toate datele și metadatele - detectează dacă obiectul este cel corect și dacă a fost modificat.

• Stocarea terțiară este de obicei construită folosind suporturi media detașabile - poate fi WORM sau doar pentru citire, tratată ca discuri fixe.

• Discul fix este de obicei mai fiabil decât discul detașabil sau unitatea de bandă.

• Memoria principală este mult mai scumpă decât stocarea pe disc.

**Capitolul 11 - Interfața Sistemului de Fișiere**

• Fișier – Vedere logică uniformă a stocării informațiilor (indiferent de suportul fizic)

◦ Mapează pe dispozitive fizice (de obicei, nevolatil)

◦ Cea mai mică alocare de stocare care poate fi denumită

◦ Tipuri: Date (numerice, de caractere, binare), Program, Formă liberă, Structurat

◦ Structura este decisă de OS și/sau program/programator (structura de date abstract)

• Atribute:

◦ Nume: Singura informație în formă lizibilă de către om

◦ Identificator: Etichetă unică, identifică fișierul în cadrul sistemului de fișiere

◦ Tip, Dimensiune

◦ Locație: pointer către locația fișierului

◦ Timp, dată, identificare utilizator

• Operațiuni: creare, scriere, citire, repoziționare în fișier, ștergere, truncare

• Tabel global menținut care conține informații independente de proces despre fișierele deschise: tabelul fișierelor deschise

◦ Tabelul fișierelor deschise per proces conține informații relevante, plus pointer către intrarea în tabelul global al fișierelor deschise

• Blocarea fișierelor deschise: mediază accesul la un fișier (partajat sau exclusiv)

◦ Obligatorie – acces refuzat în funcție de blocările deținute și cele solicitate

◦ Consultativă – procesul poate afla starea blocărilor și decide ce să facă

• Tipul de fișier poate indica structura internă a fișierului

• Metode de acces: Acces secvențial, acces direct

◦ Acces secvențial: modelul bandă al unui fișier

◦ Acces direct: acces aleatoriu, acces relativ

• Diskul poate fi subdivizat în partiții; discurile sau partițiile pot fi protejate împotriva eșecurilor cu RAID.

◦ Poate fi utilizat brut fără un sistem de fișiere sau formatat cu un sistem de fișiere

◦ Partițiile sunt cunoscute și ca minidiscuri, slice-uri

• Volumul conține sistemul de fișiere: urmărește, de asemenea, informațiile sistemului de fișiere în directorul dispozitivului sau în cuprinsul volumului

• Sistemul de fișiere poate fi general sau specializat. Unele sisteme de fișiere specializate:

◦ tmpfs – sistem de fișiere temporar în memorie volatilă

◦ objfs – sistem de fișiere virtual care oferă debuggerelor acces la simbolurile kernelului

◦ ctfs – sist de fișiere virtual care menține informații pt a gestiona procesele care încep la pornirea sistemului

◦ lofs – sistem de fișiere cu buclă inversă permite accesarea unui sistem de fișiere în locul altuia

◦ procfs – sistem de fișiere virtual care prezintă informații despre toate procesele sub forma unui sist de fișiere

• Directorul este similar unui tabel de simboluri – traduce numele de fișiere în intrările lor de director

◦ Ar trebui să fie eficient, convenabil pentru utilizatori, cu grupare logică

◦ Cel mai popular este structurat sub formă de arbore – permite gruparea

◦ Comenzi pentru manipulare: ștergere – rm <nume-fișier>; creare director nou - mkdir <nume-dir>

• Directorul curent: locația implicită pentru activități - se poate specifica și o cale pt a efectua activități în ac locație

• Directoarele cu grafic aciclic adaugă capacitatea de a partaja directoare între utilizatori

◦ Aciclicitatea poate fi garantată prin: permiterea doar a fișierelor partajate, nu a subdirectoarelor partajate; colectare de gunoi; smecanism de verificare dacă noile legături sunt OK

• Sistemul de fișiere trebuie montat înainte de a putea fi accesat – o structură de date din kernel păstrează urmărirea punctelor de montare

• Într-un sistem de partajare a fișierelor, ID-urile utilizatorului și ID-urile de grup ajută la identificarea permisiunilor unui utilizator

• Modelul client-server permite mai multor clienți să monteze sisteme de fișiere la distanță de la servere – NFS (UNIX), CIFS (Windows)

• Semantica de consistență specifică modul în care mai mulți utilizatori accesează simultan un fișier partajat-similar cu algoritmii de sincronizare din Capitolul 7

◦ O modalitate de protecție este Accesul Controlat: când un fișier este creat, se determină accesul r/w/x pentru utilizatori/grupuri

**Capitolul 12 - Implementarea Sistemului de Fișiere**

• Sistemul de fișiere se află pe stocarea secundară - discuri; sistemul de fișiere este organizat în straturi →

• Blocul de control al fișierului: structură de stocare care conține informații despre un fișier (există per fișier)

• Driverul dispozitivului: controlează dispozitivul fizic; gestionează dispozitivele I/O

• Modulul de organizare a fișierelor: înțelege fișierele, adresele logice și blocurile fizice

◦ Traduce numărul blocului logic în numărul blocului fizic

◦ Gestionarea spațiului liber, alocarea pe disc

• Sistemul de fișiere logic: gestionează informațiile de metadate - menține blocurile de control ale fișierelor

• Blocul de control de pornire: conține informațiile necesare sistemului pentru a porni OS de pe volum

• Blocul de control al volumului: conține detalii despre volum; de exemplu: număr total de blocuri, număr de blocuri libere, dimensiunea blocului, indicatoare de blocuri libere

• Partiția rădăcină: conține OS; montată la pornire

◦ Pentru toate partițiile, sistemul este verificat în mod consistent la momentul montării

◦ Verificați metadatele pentru corectitudine - se permite montarea numai dacă aceasta este îndeplinită

• Sistemele de fișiere virtuale oferă o modalitate orientată pe obiect de implementare a sistemelor de fișiere

• Directoarele pot fi implementate sub formă de liste liniare sau tabele hash

◦ Listă liniară de nume de fișiere cu pointer către blocurile de date - simplu, dar lent

◦ Tabel hash - listă liniară cu o structură de date hash - reduce timpul de căutare

▪ Bun dacă intrările au dimensiuni fixe

▪ Coliziuni pot apărea în tabelele hash atunci când două nume de fișiere au aceeași valoare hash

• Alocare contiguă: fiecare fișier ocupă un set de blocuri contiguu

◦ Simplu, cea mai bună performanță în majoritatea cazurilor; problemă - găsirea spațiului pentru fișier, fragmentare externă

◦ Sistemele de fișiere bazate pe extent sunt scheme de alocare contiguă modificate - un extent este alocat pentru alocarea fișierului

• Alocarea legată: fiecare fișier este o listă legată de blocuri - nicio fragmentare externă

◦ Localizarea unui bloc poate dura multe I/O-uri și căutări pe disc

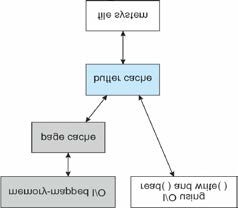
• Alocare indexată: fiecare fișier are propriul bloc de index(i) cu pointer(e) către blocurile sale de date

◦ Necesită tabel de index; poate fi acces aleatoriu; acces dinamic fără fragmentare externă, dar are costuri suplimentare

• Cele mai bune metode: legată este bună pentru secvențial, nu și pentru aleatoriu; contiguă este bună pentru secvențial și aleatoriu

• Sistemul de fișiere menține o listă de spațiu liber pentru a urmări blocurile/clusterele disponibile

• Vectorul de biți sau harta de biți (n blocuri): calculul numărului blocului → (#biți/cuvânt)\*(#cuvinte cu valoare 0)+(deplasarea pentru primul bit)

 ◦ Exemplu:

disk size = 240 bytes (1 terabyte) *n* = 240/212 = 228 bits (or 256 MB) if clusters of 4 blocks -> 64MB of memory

block size = 4KB = 212 bytes

• Hărțile spațiului (folosite în ZFS) împart spațiul dispozitivului în unități metaslab și gestionează metaslab-urile

◦ Fiecare metaslab are o hartă spațială asociată

• Cache-ul tampon - secțiune separată a memoriei principale pentru blocurile utilizate frecvent

• Scrierile sincrone sunt uneori solicitate de aplicații sau necesare de OS - fără memorie tamponare

◦ Scrierile asincrone sunt mai comune, pot fi tamponate, mai rapide

• Tehnici de optimizare a accesului secvențial: free-behind și read-ahead

• Cache-ul paginii stochează pagini în loc de blocuri de disc utilizând tehnici și adrese de memorie virtuală

◦ I/O bazat pe memorie utilizează cache-ul paginii, în timp ce I/O-ul obișnuit prin sistemul de fișiere utilizează cache-ul tampon (de disc)

• Cache-ul tamponului unificat: utilizează aceeași pagină de cache pentru a memora atât paginile cu memorie mapată, cât și I/O-ul obișnuit al sistemului de fișiere pentru a evita dubla memorie tamponare

**Capitolul 13 - Sistemele de I/O**

• Driverele dispozitivelor încapsulează detaliile dispozitivului - prezintă o interfață uniformă de acces la dispozitive pentru subsistemul I/O

• Port: punct de conexiune pentru dispozitiv

• Bus: lanț în serie sau acces direct comun

• Controller (adaptator gazdă): electronica care operează portul, busul, dispozitivul - uneori integrată

◦ Conține procesor, microcod, memorie privată, controlor de bus

• I/O bazat pe memorie: datele și registrele de comandă ale dispozitivului sunt mapate la spațiul de adrese al procesorului

◦ În special pentru spațiile de adrese mari (grafică)

• Așteptarea pentru fiecare byte de date - așteptare activă pentru I/O de la dispozitiv

◦ Rațional pentru dispozitive rapide, ineficient pentru cele lente

◦ Poate avea loc în 3 cicluri de instrucțiuni

• Linia de cerere a întreruperii CPU este declanșată de dispozitivele I/O - manipularea întreruperii

◦ Manipulatorul poate fi mascat pentru a ignora sau întârzia unele întreruperi

◦ Vectorul de întreruperi dirijează întreruperea către manipulatorul corect - în funcție de prioritate; unele pot fi nemascat

◦ Înlănțuirea întreruperilor apare dacă există mai mult de un dispozitiv la aceeași număr de întrerupere

◦ Mecanismul de întreruperi este folosit și pentru excepții

• Accesul direct la memorie este utilizat pentru a evita I/O-ul programat pentru deplasarea mare a datelor

◦ Necesită controlor DMA

◦ Ocolește CPU-ul pentru a transfera date direct între dispozitivul I/O și memorie

• Stratul de driver al dispozitivului ascunde diferențele dintre controlerele I/O de la kernel

• Dispozitivele variază în multe dimensiuni: flux de caractere/bloc, acces secvențial/aleator, sincron/asincron, partajat/dedicat, viteză, rw/ro/wo

• Dispozitivele bloc includ unitățile de discuri: I/O brut, I/O direct

◦ Comenzile includ citirea, scrierea, căutarea

• Dispozitivele de caractere includ tastaturi, șoareci, porturi seriale

◦ Comenzile includ get(), put()

• Dispozitivele de rețea au, de asemenea, propria lor interfață; UNIX și Windows NT/9x/2000 includ interfața pentru socket

◦ Abordările includ conducte (pipes), FIFO-uri, fluxuri (streams), cozi, cutii poștale (mailboxes)

• Temporizatorul intervalului programabil: folosit pentru cronometrare, întreruperi periodice

• I/O blocant: procesul este suspendat până când I/O este finalizat - ușor de utilizat și de înțeles, nu întotdeauna metoda cea mai bună

• I/O neblocant: apelul I/O returnează cât este disponibil - implementat prin multithreading, returnează rapid

• Asincron: procesul rulează în timp ce I/O se execută - dificil de utilizat, procesul este semnalat la finalizarea I/O

• Încărcarea tamponului: menținerea ieșirii pentru un dispozitiv - dacă dispozitivul poate servi doar o cerere odată (ex: imprimantă)

• Rezervarea dispozitivului: oferă acces exclusiv la un dispozitiv - trebuie să fie atent pentru a evita blocajele

• Kernelul menține informații despre starea componentelor I/O, inclusiv tabelele de fișiere deschise, conexiunile de rețea, stările dispozitivelor de caractere

◦ Structuri de date complexe urmăresc bufferele, alocarea memoriei, blocurile "nesalvate"

• STREAM: canal de comunicare full-duplex între procesul la nivel de utilizator și dispozitiv în UNIX

◦ Fiecare modul conține o coadă de citire și o coadă de scriere

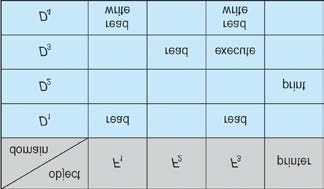
◦ Comunicarea între cozi se realizează prin transmiterea de mesaje - opțiunea de control al fluxului indică disponibilitatea sau ocuparea

◦ Asincron în intern, sincron acolo unde procesul utilizator comunică cu capul fluxului

• I/O este un factor major în performanța sistemului - solicitare asupra CPU-ului, schimbarea contextului, copierea datelor, traficul de rețea

**Capitolul 14 - Protecție**

• Principiul celui mai mic privilegiu: programele, utilizatorii, sistemele ar trebui să primească doar suficiente privilegii pentru a-și îndeplini sarcinile

• Drept de acces = <nume-obiect, set-de-drepturi> cu set-de-drepturi submulțime a tuturor operațiilor valide realizabile asupra obiectului

◦ Domeniu: mulțime de drepturi de acces

▪ Sistemul UNIX constă din 2 domenii: utilizator, supervisor

▪ Implementarea domeniilor MULTICS (inele de domenii) - dacă j<i → Di Dj

• Matricea de acces: rândurile reprezintă domeniile, coloanele reprezintă obiectele

◦ Acces(i,j) este mulțimea de operații pe care un proces care rulează în Domeniui poate invoca pe Obiectulj

◦ Poate fi extinsă pentru protecția dinamică

• Designul matricei de acces separă mecanismul de politică

◦ Mecanism: SO furnizează matricea de acces și regulile - se asigură că matricea este manipulată doar de utilizatorii autorizați

◦ Politica: Utilizatorul dictează politica - cine poate accesa ce obiect și în ce mod

• Solaris 10 utilizează controlul accesului bazat pe roluri (RBAC) pentru a implementa cel mai mic privilegiu

• Revocarea drepturilor de acces

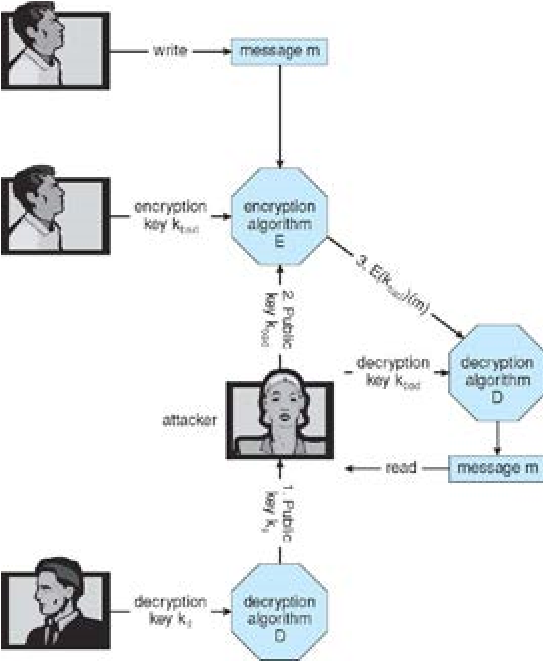
◦ Listă de acces: șterge drepturile de acces din lista de acces - simplu, imediat

◦ Listă de capabilități: necesară pentru a localiza capabilitatea în sistem înainte ca aceasta să poată fi retrasă - reachiziționare, indicatoare înapoi, indirecții, chei

• Protecția bazată pe limbaj: permite o descriere la un nivel înalt a politicilor pentru alocarea și utilizarea resurselor

◦ Poate furniza software pentru implementarea protecției atunci când verificarea suportată de hardware nu este disponibilă

**Capitolul 15 - Securitate**

• Un sistem este securizat atunci când resursele sunt utilizate și accesate conform intențiilor în toate circumstanțele

• Atacurile pot fi accidentale sau malitioase

◦ Mai ușor de protejat împotriva utilizării accidentale decât a abuzului malitios

• Categorii de încălcări de securitate:

◦ Încălcare a confidențialității - citirea neautorizată a datelor

◦ Încălcare a integrității - modificarea neautorizată a datelor

◦ Încălcare a disponibilității - distrugerea neautorizată a datelor

◦ Furt de servicii - utilizarea neautorizată a resurselor

◦ Negarea serviciului - împiedicarea utilizării legitime

• Metode de încălcare:

◦ Masquerading - pretinderea că sunt un utilizator autorizat

◦ Omul din mijloc - intrusul se află în fluxul de date, pretinzând a fi expeditorul către destinatar și viceversa

◦ Răpirea sesiunii - interceptarea unei sesiuni deja stabilite pentru a ocoli autentificarea

• Securitatea eficientă trebuie să aibă loc la patru nivele: fizic, uman, sistem de operare, rețea

• Amenințări de program: cal troian (spyware, pop-up, etc.), ușă secretă, bombă logică, depășirea stivei și a bufferului

• Viruși: fragment de cod încorporat într-un program legitim; autoreplicare

◦ Specific arhitecturii CPU, SO, aplicații

◦ Carrier de virus: introduce virusul în sistem

• Windows este ținta majorității atacurilor - cel mai comun, toată lumea este administrator

• Viermi: utilizează mecanismul de spawn - program autonom

• Scanare de porturi: încercare automată de conectare la o gamă de porturi pe o singură adresă IP sau o gamă de adrese IP

◦ Lansată frecvent de pe sisteme zombi pentru a reduce urmărirea

• Negarea serviciului: suprasarcină a computerului țintă pentru a împiedica desfășurarea activităților utile

• Criptografia: mijloc de a restrânge potențialii expeditori și/sau destinatari - bazată pe chei

◦ Permite confirmarea sursei, primirea de către destinația specificată, relația de încredere

• Criptarea: [K de chei], [M de mesaje], [C de texte cifrate], funcția E:K pentru criptare, funcția D:K pentru decriptare

◦ Poate fi simetrică și asimetrică (distribuie cheia publică de criptare, deține cheia privată de decriptare)

▪ Asimetrica necesită mult mai multă putere de calcul - nu este folosită pentru tranzacții de date în masă

▪ Cheile pot fi stocate pe un inel de chei

• Autentificare: restrângerea unui set de potențiali expeditori ai unui mesaj

◦ Ajută la dovedirea faptului că mesajul nu a fost modificat

◦ Funcțiile de dispersie stau la baza autentificării

▪ Creează un bloc mic și fix de date (rezumat de mesaj, valoare de dispersie)

• Criptarea simetrică este utilizată în codul de autentificare a mesajelor (MAC)

• Autentificatorii produși de algoritmul de autentificare sunt semnături digitale

• Autentificarea necesită mai puține calcule decât metodele de criptare

• Certificatele digitale: dovada cui sau ce aparține o cheie publică

• Apărare în adâncime: teoria de securitate cea mai comună - multiple straturi de securitate

• Se poate încerca detectarea intruziunii:

◦ Bazată pe semnături: detectează "modele rele"

◦ Detectarea anomaliilor: detectează diferențe față de comportamentul normal

▪ Ambele pot raporta rezultate fals pozitive sau fals negative

◦ Auditarea, contabilitatea și jurnalizarea activității specifice a sistemului sau rețelei