UNIVERSITATEA *TRANSILVANIA* DIN BRA□OV
Centrul de Învățământ la Distanță
și Învățământ cu Frecvență Redusă

FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
PROGRAM DE LICENȚĂ: INFORMATICĂ

Sisteme de operare

CURS PENTRU ÎNVĂȚĂMÂNT LA DISTANȚĂ

AUTORI: **Conf. Dr. Ion Florea**

ANUL I

2012-2013

CUPRINS

Introducere

Sistemul de operare reprezintă componenta software fundamentală a unui sistem de calcul. Sistemul de operare permite aplicațiilor şi utilizatorilor să acceseze resursele unui sistem de calcul. De asemenea, sistemul de operare controlează modul cum sunt distribuite resursele sistemului, încercând să optimizeze accesarea lor şi să ofere un grad ridicat de protecție al acestora. Un alt scop al unui sistem de operare este să asigure flexibilitate în funcționarea calculatorului pe care este instalat. Sistemele de operare permit interacțiunea sistemului de calcul cu mai mulți utilizatori, execuția simultană a mai multor aplicații, precum şi adaptarea strategiilor de răspuns la o anumită problemă.

Comparativ, dacă privim un calculator ca un aeroport, atunci sistemul de operare poate fi privit ca turnul de control care controlează traficul. Resursele aeroportului sunt reprezentate de pistele şi avioanele pe care le pune la dispoziție. Fără un turn de control care să asigure gestiunea şi organizarea acestor resurse, pe un aeroport nu s-ar putea evita întârzierile şi coliziunile. În cazul sistemului de calcul, rolul sistemului de operare este de a coordona traficul şi accesul la resurse, ca şi turnul de control al aeropertului. De multe ori, se spune că dacă unitatea centrală a calculatorului este „creierul", atunci sistemul de operare este „inima" acestuia.

Obiectivele cursului

Cursul intitulat *Sisteme de operare-Concepte teoretice şi studii de caz* are ca obiectiv principal însuşirea conceptelor fundamentale pe care se sprijină orice sistem de operare. După parcurgerea acestui curs, studenții îşi vor schimba viziunea despre ceea ce înseamnă un sistem de calcul. De asemenea, vor obține abilități în utilizarea sistemelor de operare cele mai utilizate, cum sunt cele din familiile Windows şi Linux. La sfârşitul acestui curs, studentul va fi capabil să:

- opereze cu noțiuni precum: administrarea proceselor, gestiunea memoriei interne, gestiunea fişierelor, administrarea intrărilor şi ieşirilor;
- explice modalitățile de implementare ale mecanismelor fundamentale în cazul sistemelor Windows şi Linux;
- să utilizeze interfețele în mod grafic şi text oferite de sistemele Windows şi Linux.

Cerințe preliminare.

Studenții trebuie să posede cunoştințe, deprinderi, abilități obținute în cadrul cursurilor de Algoritmică şi Arhitectura calculatoarelor.

Resurse.

Sisteme de calcul pe care să fie instalate:

- un sistem de operare din familia Windows, respectiv Linux ;
- medii de programare tradiționale (C++, Java);

Structura cursului

Volumul este structurat în şase module, care sunt formate din unități de învățare. La rândul său, fiecare unitate de învățare cuprinde: obiective, aspecte teoretice privind tematica unității de învățare respective, exemple, teste de evaluare precum

şi probleme propuse spre discuție şi rezolvare.

Primul modul cuprinde o unitate de învățare, prin care cititorul este introdus în problematica sistemelor de operare. În acest context, este prezentată evoluția sistemelor de operare, strâns legată de evoluția sistemelor de calcul. De asemenea, se încearcă o definire a conceptului de sistem de operare şi stabilirea locului acestuia în cadrul unui sistem de calcul. Sunt prezentate conceptele fundamentale cu care lucrează sistemele de operare si legatura dintre sistemul de operare si componenta hardware a sistemului de calcul.

Al doilea modul abordează problematica gestiunii proceselor şi cuprinde trei unități de învățare. Prima unitate de învățare prezintă coordonarea execuției proceselor care, în condițiile multiprogramării reprezintă una dintre problemele esențiale pe care trebuie să le rezolve sistemul de operare. A doua unitate de învățare abordează problematica planificării execuției proceselor, prin care se realizează o servire echitabilă a aplicațiilor de către unitatea centrală de prelucrare. Rezolvarea problemei interblocării proceselor este esențială în condițiile în care acestea partajează diverse resurse ale siatemelor de calcul, fiind abordată în a treia unitate de învățare.

Al treilea modul abordează problematica gestiunii memoriei interne a sistemelor de calcul. Memoria internă este una dintre componentele fizice esențiale ale calculatoarelor şi administrarea ei corectă reprezintă una dintre sarcinile fundamentale ale sistemelor de operare. Prima unitate de învățare prezintă metodele clasice de alocare a memoriei, specifice primelor generații de calculatoare. A doua unitate de învățare abordează metodele de alocare a memoriei care se întâlnesc în cadrul calculatoarelor moderne.

Fişierul reprezintă unul dintre conceptele fundamentale ale sistemului de operare, fiind esențial în păstrarea şi manipularea datelor. Administrarea fişierelor reprezintă tematica celui de al patrulea modul de învățare.

Al cincelea modul abordează problematica interacțiunii dintre sistemul de operare şi dispozitivele de intrare şi ieşire. Această componentă a sistemului de operare trebuie să transmită comenzi către acestea, să trateze eventuale întreruperi sau erori. De asemenea, sistemul de operare trebuie să furnizeze o interfață simplă, uşor de utilizat de către utilizator. În acest sens, controlul perifericelor unui calculator este o preocupare importantă a proiectanților de sisteme de operare.
Modulul al şaselea prezintă modalitățile prin care sunt aplicate conceptele prezentate în cadrul modulelor anterioare în cazul celor mai răspândite sisteme de operare, cele din familiile Linux şi Windows. Modulul este format din două unități de învățare, corespunzătoare celor două clase de sisteme de operare.

Evaluarea va cuprinde atât verificarea însuşirii cunoştiințelor teoretice prezentate, cât şi utilizarea practică a sistemelor de operare.

Modulul 1. Concepte introductive despre sisteme de operare

Cuprins

M1.U1.1. Introducere Dacă unitatea centrală de prelucrare (CPU) reprezintă "creierul" sistemului de calcul, sistemul de operare (SO) reprezintă „inima" calculatorului. Atunci când un utilizator sau, mai concret o aplicație solicită o resursă a sistemului de calcul, aceasta este oferită prin intermediul sistemului de operare. Cererea respectivă se face printr-o instrucțiune (comandă), care este analizată de CPU şi care transmite un mesaj către SO prin care se specifică resursa cerută şi condițiile în care aceasta va fi utilizată.

Este dificil să se dea o definiție completă a ceea ce este un SO, în schimb, este mult mai uşor să se vadă ce face un SO. Astfel SO oferă facilitățile necesare unui programator pentru a-şi construi propriile aplicații. De asemenea, SO gestionează resursele fizice(memorie, discuri, imprimante etc) şi cele logice (aplicații de sistem, fişiere, baze de date etc) ale sistemului de calcul, oferind posibilitatea ca utilizatorii să poată folosi în comun aceste resurse. SO oferă o interfață prin care aplicațiile utilizator şi cele de sistem au acces la componenta hardware. SO oferă un limbaj de comandă prin care utilizatorii pot să-şi lanseze în execuție propriile programe şi să-şi manipuleze propriile fişiere.

Putem spune că sistemul de operare realizează două funcții de bază: extensia funcțională a calculatorului şi gestiunea resurselor.

M1.U1.2. Obiectivele unității de învățare

Această unitate de învățare îşi propune ca obiectiv principal o introducere a studenților în problematica sistemelor de operare. La sfârşitul acestei unități de învățare studenții vor fi capabili să:

- înțeleagă şi să explice locul sistemului de operare în cadrul sistemului de calcul;
- înțeleagă şi să explice evoluția sistemelor de operare.
- înțeleagă şi să explice legătura între sistemul de operare şi componenta hardware a sistemului de calcul;
- înțeleagă şi să explice tipurile de sisteme de operare;
- înțeleagă şi să explice încărcarea sistemului de operare.

Durata medie de parcurgere a unității de învățare este de 2 ore.

M1.U1.3. Definirea sistemului de operare

Componentele principale ale unui sistem de calcul (SC) sunt cea fizică(hardware) şi cea logică(software). La rândul ei, componenta logică este formată din aplicații şi sistemul de operare-SO (figura 1.1). Orice utilizator accesează diverse aplicații, prin intermediul unei interfețe care este oferită tot de sistemului de operare.

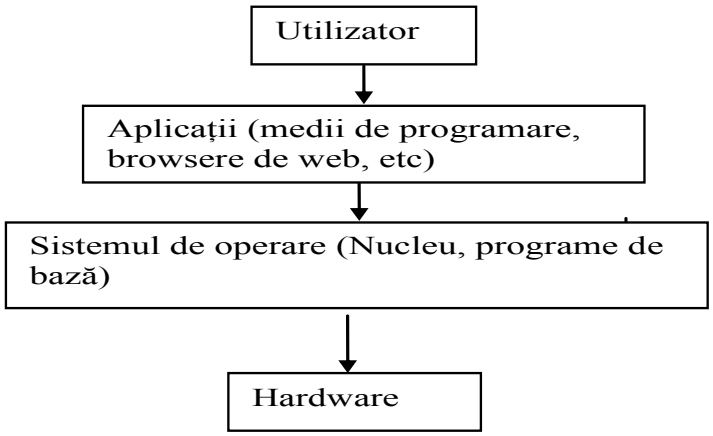


Figura 1.1 Rolul unui sistem de operare

Sistemul de operare este format din două componente: nucleu şi programe de bază. **Nucleul** este componenta de bază a sistemului de operare care are rolul de a gestiona resursele sistemului de calcul şi de a oferi o interfață aplicațiilor către acestea. O descriere detaliată este prezentată în cadrul acestei unități de învățare. **Programele de bază** sunt aplicațiile fundamentale ale unui sistem de operare, care permit interacțiunea cu nucleul şi cu componentele fizice ale sistemului de calcul (interpretoare de comenzi, utilitare de gestiune a utilizatorilor şi a fişierelor, biblioteci etc.).

Funcțiile de bază ale sistemelor de operare. Aceste funcții cooperează între ele pentru a satisface cerințele utilizatorilor. În figura 1.2 sunt prezentate interacțiunile între modulele care realizează funcțiile SO, precum şi între aceste module şi componentele hard ale sistemului de calcul.

Administrarea resurselor hardware se referă la modul de acces la procesor, memorie, hard discuri, comunicare în rețea precum şi la dispozitivele de intrare/ieşire a diverselor aplicații care se execută, de sistem sau utilizator. Aceste componente hardware sunt alocate, protejate în timpul alocării şi partajate după anumite reguli. Sistemul de operare oferă o abstractizare a resurselor hardware, prin care se oferă o interfață simplificată la acestea. Dispozitivele fizice, fişierele şi chiar sistemul de calcul au nume logice prin care pot fi accesate de utilizator sau diverse aplicații. Accesarea acestor componente se realizează prin intermediul driverelor. Această problemă va fi discutată pe larg în modulul V.

Administrarea proceselor şi a resurselor. Procesul reprezintă unitatea de bază a calculului, definită de un programator, iar resursele sunt elemente ale mediului de calcul necesare unui proces pentru a fi executat. Crearea, execuția şi distrugerea proceselor, comunicarea între ele,

împreună cu alocarea resurselor după anumite politici, sunt aspecte deosebit de importante care vor fi discutate în detaliu în modulul II.

Gestiunea memoriei. SO alocă necesarul de memorie internă solicitat de procese şi asigură protecţia memoriei între procese. O parte este realizată prin hard, iar o parte prin soft. Această problemă va fi obiectul modulul III.

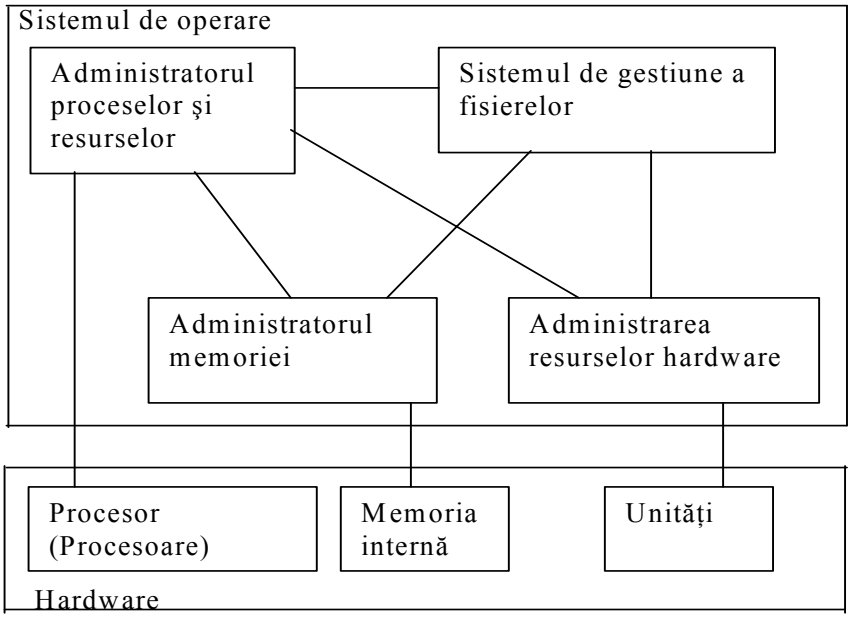


Figura 1.2. Organizarea sistemului de operare

Gestiunea fişerelor. SO conţine o colecţie de module prin intermediul cărora se asigură deschiderea, închiderea şi accesul utilizatorului la fişierele rezidente pe diferite suporturi de informaţii. Componentă de bază a SO, este cel mai des invocată de către utilizator şi de către operator. Ea va fi subiectul modulul IV.

M1.U1.4. Evoluţia sistemelor de operare

Primele calculatoare apărute(**generaţia I**) dispun numai de echipamentul hard. Ele puteau fi programate numai în cod maşină, care este introdus în memorie de la o consolă. După execuţie, se vizualizează rezultatele şi se corectează , eventuale erori. Alocarea timpului de lucru în sistem se realiza manual; fiecare utilizator primea un anumit interval de timp pentru a-şi executa programele.

Generaţia a II-a de calculatoare a durat între anii 1955-1965 şi este legată de utilizarea tranzistoarelor, la producerea sistemelor de calcul. Apar o serie de componente hardware noi: cititorul de cartele, imprimanta, banda magnetică etc. În ceea ce priveşte componenta software, principalele aspecte sunt:
- Apare pentu prima dată un compilator pentru un limbaj evoluat (FORTRAN).
- Este realizat **monitorul rezident**, ce reprezintă o formă rudimentară de sistem de operare; datele, textul sursă şi informaţiile destinate monitorului erau citite de pe cartelele perforate. Acesta realiza atât înlănţuirea fazelor **execuţiei** programului (**compilarea, editarea legăturilor**, **execuţia** programului), cât şi cea a lucrărilor (**job-urilor).** Un job era format din unul sau mai multe programe**.** Între cartelele care conţin textul sursă al programului şi datele de prelucrat, se inserau cartele speciale numite **cartele de comandă,** care se adresează acestui monitor rezident. Astfel, s-a definit un **limbaj de control** al job-urilor**.** Prin intermediul lui, se comandă trecerea de la o fază la alta sau de la un job la altul.

- Un astfel de mod de prelucrare se numeşte **serial,** deoarece job-urile se execută unul după altul sau **prelucrare pe loturi** (**Batch processing**), adică se execută mai multe job-uri unul după altul.
- O caracteristică a acestui sistem este **monoprogramarea**, adică CPU nu se ocupă de alt job până când nu-l termină pe cel curent.
- Deoarece lucrul cu cartele perforate este incomod, conținutul acestora era transferat pe benzi magnetice, folosind calculatoare relativ ieftine. Conținutul benzilor era utilizat de către sistemul de calcul în locul cartelelor perforate, unitatea de bandă fiind intrarea standard. Analog, rezultatele, sub forma unor linii de imprimantă, sunt depuse mai intâi într-un fisier pe un suport magnetic, iar la terminarea lucrului, conținutul fişierului este listat la imprimantă. Prin această metodă, se reduce şi timpul de lenevire al CPU datorat vitezei de lucru mai mici a unor periferice.
-Apare şi noțiunea de independență față de dispozitiv a programelor, adică aceeaşi operație de I/O să poată fi realizată de pe diferite dispozitive fizice. Acest lucru se realizează prin aşa zisele dispozitive logice de I/O, ce reprezintă nişte identificatori utilizați de programe, care sunt asociați prin intermediul SO dispozitivelor fizice.

Generația a III-a de calculatoare apare în 1965 şi este legată de utilizarea circuitelor integrate. Din punct de vedere hardware, principala componentă apărută este hard disc-ul. Principalele aspecte legate de sistemul de operare sunt:
- Apare conceptul de **multiprogramare,** adică execuția în paralel a mai multor job-uri. Astfel, se asigură un grad de utilizare mai ridicat al CPU, deoarece atunci când un job execută o operație de intrare-ieşire, CPU poate servi alt job şi nu aşteaptă terminarea acestuia.
- Apariția discurilor magnetice, cu acces direct la fişiere a permis operarea on-line, simultan, cu mai multe periferice (SPOOLING - Simultaneous Peripherical Operation On-Line). El s-a obținut prin îmbinarea utilizării zonelor tampon multiple cu conversiile off-line şi cu multiprogramarea.
- Un alt concept fundamental al acestei generații este de utilizare în comun a timpului CPU (timesharing). Conform acestui principiu, mai mulți utilizatori pot folosi simultan terminale conectate la acelaşi calculator. Se elimină astfel o parte din timpii morți datorați diferenței dintre viteza de execuție a CPU şi cea de reacție a utilizatorilor.

Exemple Dintre sistemele de operare specifice acestei generații de calculatoare, cele mai reprezentative sunt sistemul SIRIS pentru calculatoarele IRIS 50, care s-au produs şi în țara noastră sub denumirea FELIX 256/512/1024 şi sistemul OS/360 pentru calculatoarele IBM 360.

Generației a-IV-a de calculatoare a apărut în anul 1980 şi este legată de utilizarea tehnologiei microprocesoarelor. Principalele caracteristici ale acestei generații sunt**:**
- Interactivitatea. Un **sistem interactiv** permite **comunicarea on-line** între utilizator şi sistem. Utilizatorul are la dispoziție un terminal cu tastatură şi ecran, prin care comunică cu sistemul. Comunicarea se poate face în mod text sau grafic. În astfel de sisteme, utilizatorul dă comanda, asteaptă răspunsul şi, în funcție de rezultatul furnizat de comanda precedentă, decide asupra noii comenzi. Spre deosebire de sistemele seriale, sistemele interactive au un timp de răpuns rezonabil de ordinul secundelor.
- Extinderea conceptului de **time-sharing** prin îmbinarea cu **interactivitatea** şi **multiprogramarea**.
- **Terminal virtual** este modalitatea prin care utilizatorul stabileşte o sesiune cu sistemul, fiind o simulare a hardware-ului calculatorului şi implementat de către sistemul de operare. Sistemul comută rapid de la un program la altul, înregistrând comenzile solicitate de fiecare utilizator prin terminalul său. Deoarece o tranzacție a utilizatorului cu sistemul necesită un timp de lucru mic al CPU, rezultă că într-un timp scurt, fiecare utilizator este servit cel puțin o

dată. În acest fel, fiecare utilizator are impresia că lucrează singur cu sistemul. Dacă sistemele seriale încearcă să optimizeze numărul de job-uri prelucrate pe unitatea de timp, sistemele timesharing realizează o servire echitabilă a mai multor utilizatori, aflați la diverse terminale.
- Odată cu evoluția sistemelor timesharing, s-a realizat o diferențiere între noțiunile de job şi proces (program în execuție). În sistemele timesharing, la acelaşi moment un job poate executa două sau mai multe procese, pe când în sistemele seriale un job presupune un singur proces. Într-un sistem timesharing multiprogramat procesele se mai numesc şi **task**-uri iar un astfel de calculator se mai numeşte şi sistem multitasking.

- Redirectarea si **legarea in pipe**. Aplicarea lor presupune că fiecare program lansat de la un terminal are un **fişier standard de intrare** şi un **fişier standard de ieşire.** De cele mai multe ori acestea coincid.cu tastatura, respectiv cu terminalul de la care se fac lansările. Redirectarea intrărilor standard (intrare sau/şi ieşire) permite utilizatorului să înlocuiască intrarea standard cu orice fişier, respective să se scrie rezultatele, afişate de obicei pe ecran într-un fişier oarecare, nou creat sau să fie adăugate la un fişier deja existent. Informațiile de redirectare sunt valabile din momentul lansării programului pentru care se cere acest lucru şi până la terminarea lui. Dupa terminarea programului se revine la fişierele standard implicite.

Exemplu. Dintre SO cele mai cunoscute ale generației a IV-a de calculatoare, amintim sistemele DOS, Windows, Unix şi Linux pentru calculatoare personale şi pentru rețele de calculatoare, sistemele RSX pentru minicalculatoare (calculatoare mainframe).

M1.U1.5. Tipuri de sisteme de operare

Sisteme de operare pentru calculatoare mari. Calculatoarele mari sunt caracterizate de capacitatea de a procesa volume foarte mari de informatii. Ele lucrează cu dispozitive de I/O de capacitate mare de stocare şi sunt orientate spre execuția mai multor sarcini în acelaşi timp. Pe astfel de calculatoare se pot executa: servere de Web, servere pentru site-uri dedicate comerțului electronic şi servere care gestionează tranzacții între companii. Aceste sisteme de operare oferă trei tipuri de servicii: procesarea loturilor, procesarea tranzacțiilor şi partajarea timpului.

Un sistem de procesare în loturi prelucrează lucrări obişnuite, care nu impun intervenția utilizatorului.

Exemple de astfel de prelucrări sunt: procesarea operațiilor uzuale din cadrul companiilor de asigurări, raportarea vânzărilor dintr-un lanț de magazine.

Sistemele de procesare a tranzacțiilor gestionează pachete mari de cereri scurte.

Exemplu de astfel de prelucrare este procesul de căutare/verificare într-o bază de date a unei companii aeriene care conține rezervările de bilete.

La un astfel de sistem de calcul, sunt legate terminale de la care se introduc un număr mare de tranzacții diverse.

Exemplu de astfel de sistem de operare este OS/370, care a fost realizat de firma IBM..

Sisteme de operare pentru calculatoare pe care se execută servere. Aceste sisteme de calcul fac parte din rețele de calculatoare şi rolul lor este de a servi cererile aplicațiilor lansate

de diverşi utilizatori. În funcție de complexitatea operațiilor pe care le execută, aceste calculatoare pot fi de la calculatoare personale la calculatoare de putere foarte mare.

Exemple de servere**:** Servere care administrează utilizatorii unei rețele de calculatoare, servere de fişiere, servere de baze de date, servere de poştă electronică, servere care asigură servicii de Web etc.

Sisteme de operare pentru calculatoare multiprocesor. Aceste sisteme de calcul încorporează mai multe unități de prelucrare (procesoare), în vederea creşterii capacității de prelucrare. Aceste calculatoare necesită sisteme de operare speciale, care să rezolve diverse probleme specifice legate de încărcarea echilibrată a procesoarelor, partajarea resurselor etc.

Sisteme de operare pentru calculatoare personale. Aceste sisteme de operare sunt concepute pentru a oferi posibilități de lucru unui singur utilizator. Ele pot funcționa individual sau legate în rețea. În cadrul unei rețele, calculatoarele pe care se execută servere folosesc variante diferite de sisteme de operare față de cele ale sistemelor de calcul personale.

Exemplu. Există patru versiuni ale sistemului Windows 2000: versiunea Profesional este destinată calculatoarelor individuale; celelalte trei versiuni (Server, Advanced Server şi DataCenter Server) sunt destinate calculatoarelor server dintr-o rețea. Windows 2000 DataCenter Server este destinat serverlor multi-procesor. Versiunea XP propune două variante: Windows XP. Net Server, utilizat de calculatoarele server dintr-o rețea şi Windows XP Proffesional, destinat calculatoarelor individuale precum şi calculatoarelor client legate în rețea.

Sisteme de operare pentru calculatoare în timp real. Sistemele de calcul în timp real consideră timpul de răspuns ca o caracteristică fundamentală a bunei lor funcționări.

Să ne reamintim...

Sistemul de operare este parte a componentei software a unui sistem de calcul. El reprezintă o extensie funcțională a calculatorului şi realizează gestiunea resurselor unui sistem de calcul.

Evoluția sistemelor de operare este legată de cea a sistemelor de calcul care se împarte în patru generații de calculatoare. Funcțiile de bază ale sistemelor de operare sunt: administrarea resurselor hardware, administrarea proceselor şi a resurselor, gestiunea memoriei şi si stemul de gestiune a fisierelor.

Înlocuiți zona punctată cu termenii corespunzători.

1. Primele calculatoare apărute **(generația I)** dispun numai de ..... Ele puteau fi programate numai în ....., care este introdus în memorie de la .....
2. Generația a II-a de calculatoare a durat între anii ..... şi este legată de utilizarea ....., la producerea sistemelor de calcul. Apar o serie de componente hardware noi: .....
3. Monitorul rezident reprezintă o formă rudimentară de ..... Acesta realiza atât înlănțuirea ....., cât şi cea a .....**.**
4. Generația a III-a de calculatoare apare în ..... şi este legată de utilizarea ..... Din punct de vedere hardware, principala componentă apărută este ..... Apare conceptul de .....**,** adică execuția în paralel a mai multor job-uri
5. Terminal virtual este modalitatea prin care utilizatorul stabileşte o sesiune cu ....., fiind o simulare a ..... calculatorului şi implementat de către .....

M1.U1.6. Organizarea hardware a sistemelor de calcul

Sistemele de calcul se bazează pe conceptul de arhitectură Von Neumann, conform căruia partea de hardware este formată din:
►Unitatea centrală de calcul (CPU - Central Processing Unit), compusă din unitatea aritmetică şi logică (ALU-Arithmetical-Logical Unit) şi unitatea de control.
►Unitatea de memorie primară sau executabilă sau internă.
►Unităţi de I/O.
Toate unităţile sunt conectate folosind o magistrală (bus), care se împarte într-o magistrală de date şi una de adrese.

Pentru memorie şi alte unităţi se poate adăuga o magistrală de I/O, care nu este folosită de CPU. O astfel de organizare, permite ca o unitate să poată citi/scrie informaţii din memorie, fără alocarea CPU. Fiecare magistrală poate fi gândită ca fiind formată din mai multe linii (fire) paralele, care pot păstra o cifră binară.

Unitatea aritmetică şi logică conţine, pe lângă unitatatea funcţională un număr de regiştri generali şi de stare. Regiştrii generali sunt folosiţi în efectuarea operaţiilor aritmetice şi logice, atât pentru memorarea operanzilor încărcaţi din memorie, cât şi a rezultatului operaţiei, care apoi va fi salvat într-o locaţie de memorie.

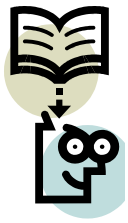
CPU extrage şi execută instrucţiunile cod maşină ale procesului încărcat în memoria internă. În acest sens, CPU conţine:
►o componentă care extrage o instrucţiune memorată într-o locaţie de memorie;
►o componentă care decodifică instrucţiunea;
►o componentă care se ocupă de execuţia instrucţiunii, împreună cu alte componente ale SC.
Regiştrii contor de program PC (Program Counter), respectiv registrul instrucţiune IR (Instrucţion Register), conţin adresa de memorie, respectiv o copie a instrucţiunii în curs de prelucrare.

Unităţile de I/O sau dispozitivele periferice sunt folosite pentru a plasa date în memoria primară şi pentru a stoca cantităţi mari de date pentru o perioadă lungă de timp. Astfel, ele pot fi unităţi de stocare (unităţi bloc), cum ar fi discurile, respectiv unităţi caracter cum ar fi tastatura, mouse-ul, display-ul terminalului precum şi unităţi de comunicaţie, cum ar fi portul serial conectat la un modem sau o interfaţă la reţea. Fiecare unitate foloseşte un controller de unitate pentru a o conecta la adresele calculatorului şi la magistrala de date. Controller-ul oferă un set de componente fizice pe care instrucţiunile CPU le pot manipula pentru a efectua operaţii de I/O. Ca şi construcţie, controller-ele diferă, dar fiecare oferă aceeaşi interfaţă de bază. Sistemul de operare ascunde aceste detalii de funcţionare ale controller-lor, oferind programatorilor funcţii abstracte pentru accesul la o unitate, scrierea/citirea de informaţii etc. Componenta sistemului de operare care manipulează dispozitivele de I/O este formată din driverele de unitate.

Moduri de lucru ale procesorului. Procesoarele contemporane conţin un bit care defineşte modul de lucru al procesorului. Acest bit poate fi setat în modul utilizator sau supervizor. În modul supervizor, procesorul poate executa orice instrucţiune cod maşină, pe când în modul utilizator el poate executa numai o parte dintre aceste instrucţiuni. Instrucţiunile care pot fi executate numai în modul supervizor se numesc instrucţiuni privilegiate. De exemplu, astfel de instrucţiuni sunt cele de I/O. Un proces, dacă este executat în mod utilizator, el nu poate să-şi execute propriile instrucţiuni de I/O. De aceea, aceste instrucţiuni sunt executate prin intermediul SO. Când un program face o cerere către sistem, o instrucţiune cod maşină specială este apelată pentru a comuta procesorul în modul supervizor şi începe să execute driverul unităţii respectiv.

Corespunzător celor două moduri de lucru, memoria internă este împărțită în zona de memorie utilizator şi zona de memorie supervizor. Dacă bitul mod de lucru este setat pe utilizator, atunci procesul respectiv are acces numai la zona cu acelaşi nume. Astfel, se realizează şi protecția zonei de memorie supervizor. **Nucleul** este acea parte a SO care este executată în modul supervizor. Alte procese, legate de diverse aplicații ale utilizatorilor sau chiar aplicații soft de sistem sunt executate în mod utilizator. Execuția acestor instrucțiuni nu afectează securitatea sistemului. Când un proces doreşte să execute anumite operații în mod supervizor, atunci se va face o comutare din modul utilizator, în cel supervizor. Acest lucru se realizează prin intermediul unei instrucțiuni **trap**, numită instrucțiune de apel al supervizorului, care setează bitul de mod de lucru şi face un salt la o locație de memorie, care se află în spațiul de memorie protejat, locație care conține începutul unei proceduri sistem care va rezolva cererea procesului. Când execuția acestei rutine supervizor s-a terminat, SO va reseta bitul de mod de lucru din supervizor în utilizator.

Să ne reamintim...



Sistemele de calcul se bazează pe conceptul de arhitectură Von Neumann, conform căruia partea de hardware este formată din:
►Unitatea centrală de calcul (CPU - Central Processing Unit), compusă din unitatea aritmetică şi logică (ALU-Arithmetical-Logical Unit) şi unitatea de control.
►Unitatea de memorie primară sau executabilă sau internă.
►Unități de I/O.
Toate unitățile sunt conectate folosind o magistrală (bus), care se împarte într-o magistrală de date şi una de adrese.

Înlocuiți zona punctată cu termenii corespunzători.
1. Unitatea centrală de calcul este compusă din ..... şi .....
2. Unitățile de ..... sau dispozitivele ..... sunt folosite pentru a plasa date în ..... şi pentru a stoca cantități ..... pentru o perioadă .....de timp.
3. Fiecare unitate foloseşte un ..... de unitate pentru a o conecta la ..... calculatorului şi la ..... de date.
4. Componenta sistemului de operare care manipulează ..... este formată din ..... de unitate.
5. Instrucțiunile care pot fi executate numai în modul ..... se numesc instrucțiuni .....

M1.U1.7. Întreruperi.

Calculatoarele furnizează un mecanism prin care diferite componente (hard sau soft) întrerup evoluția normală a procesorului, pentru a semnala acestuia diverse evenimente apărute.

Rolul principal al întreruperilor este de a îmbunătății utilizarea procesorului. Acest lucru se vede cel mai bine atunci când se execută operații de I/O. Când se lansează execuția unei astfel de operații, procesorul nu trebuie să aştepte terminarea ei; în acest timp, el poate servi un alt proces şi i se va transmite când s-a terminat de efectuat operația de intrare-ieşire printr-o întrerupere.

În cadrul unui proces, după executarea fiecărei instrucțiuni cod-maşină, CPU verifică dacă în perioada execuției acesteia a primit întreruperi. În caz afirmativ, suspendă execuția procesului respectiv şi lansează în execuție manipulatorul întreruperilor, care identifică rutina care corespunde întreruperii respective, care va fi executată în continuare. După ce execuția rutinii respective s-a terminat, se poate relua execuția procesului. Evident, că din punctul de vedere

al utilizării unității centrale, acest mecanism presupune şi el anumite „cheltuieli” ale acesteia, dar impactul asupra creşterii gradului de utilizare este semnificativ.

Exemple.
Programele pot genera întreruperi în situații de excepție (depăşire a domeniului de valori la calculul unei expresii, împîrțire la zero, încercare de execuție a unei instrucțiuni cod maşină ilegale, încercare de referențiere a unei locații de memorie la care utilizatorul nu are drept de acces etc.)

Întreruperi generate de controllerele perifericelor pentru a semnala terminarea normală a unei operații de I/O sau semnalarea unor eventuale erori apărute.

Întreruperi generate de defecțiuni hardware apărute (întrerupere a alimentării cu energie electrică, eroare de paritate a unei locații de memorie etc.).

Întreruperi generate de timer-ul procesului, prin care semnalează sistemului de operare să execute anumite funcții, la anumite momente.

M1.U1.8. Apelurile de sistem

Apelurile de sistem furnizează o interfață între un proces şi sistemul de operare. Prin intermediul acestora, un program utilizator comunică cu sistemul de operare şi cere anumite servicii de la acesta. Apelurile de sistem pot fi împărțite în cinci categorii importante:

►**controlul proceselor** (încărcarea, execuția, sfârşitul, abandonarea, setarea şi obținerea atributelor, alocarea şi eliberarea de memorie etc);

►**manipularea fişierelor** (creere, ştergere, deschidere, închidere, citire, scriere, repoziționare, setarea şi obținerea atributelor);

► **manipularea unităților** (cerere şi eliberare unitate, citire scriere şi repoziționare, obținerea şi setarea atributelor, ataşarea/detaşarea logică);

► **întreținerea informațiilor (**obținerea şi setarea timplui sau datei calendaristice sau a sistemului, obținerea de informații despre componentele fizice şi logice ale sistemului şi posibilitatea modificării lor**);**

► **comunicații** (crearea şi anularea unei conexiuni, transmiterea şi primirea de mesaje, transferul informațiilor de stare, ataşarea/detaşarea logică a unităților la distanță).

Limbajele de asamblare şi limbajele evoluate moderne conțin instrucțiuni(comenzi) prin care sunt lansate apeluri de sistem. Pentru transmiterea parametrilor, sunt utilizate trei metode:

►transmiterea parametrilor prin intermediul regiştrilor, atunci când numărul lor este relativ mic;

►transmiterea parametrilor sub forma unei tabele (bloc de memorie), a cărei adresă este pusă într-un registru;

►punerea parametrilor în vârful stivei, de unde sunt extraşi de către sistemul de operare.

După transmiterea parametrilor, este declanşată o instrucțiune **trap,** pentru oferi controlul sistemului de operare. Când sistemul de operare obține controlul, verifică corectitudinea parametrilor şi în caz afirmativ execută activitatea cerută. Când a terminat de executat rutina respectivă, sistemul de operare returnează un **cod de stare** într-un registru, care specifică terminarea normală sau anormală şi execută o instrucțiune de revenire din instrucțiunea **trap,** prin care se redă controlul procedurii de sistem, care apoi redă controlul execuției procesului care a făcut apelul, returnând anumite valori conținute în parametrii apelului.

M1.U1.9 Conceptul general de proces

Dacă sistemele de operare cu prelucrare în loturi executau lucrări(job-uri), sistemele de operare moderne, bazate pe divizarea timpului execută task-uri. Aceste două concepte corespund termenului de proces. În cadrul sistemelor de operare, procesul reprezintă o entitate activă, un program în execuție ale cărui instrucțiuni sunt parcurse secvențial şi executate de către unitatea centrală a calculatorului. Dacă prin program înțelegem o entitate statică, o codificare într-un anumit limbaj de programare a unui algoritm, sub forma unui fişier stocat pe un suport extern de informație, prin proces definim o entitate dinamică, încărcat în memoria internă a sistemului de calcul. Unul sau mai multe procese pot fi asociate unui program, dar ele sunt considerate entități distincte.

Orice proces foloseşte mai multe resurse ale sistemului de calcul: **contorul de program** (PC – **P**rogram **C**ounter) este un registru al UC care conține adresa de memorie a următoarei instrucțiuni care urmează să fie executată; **stiva de program** conține date utilizate în execuție, parametri ai subprogramelor, adrese de retur şi alte variabile locale; **secțiunea de date**, conține variabile globale; **timpul de lucru** al UC; **fişiere**; **dispozitive de I/O** etc.

Aceste resurse sunt alocate procesului fie în momentul creerii, fie în timpul execuției lui. În multe sisteme, procesul este unitatea de lucru. Astfel de sisteme constau dintr-o colecție de procese. Sistemul de operare, ca şi componentă soft, este format din mai multe procese, care execută codul de sistem, pe când procesele utilizator execută programe scrise de aceştia. Toate aceste procese se execută simultan la nivelul unui sistem de calcul sau al unei rețele de calculatoare. În mod tradițional, un proces conținea un singur fir de execuție, adică o singură instrucțiune cod maşină este executată de CPU la un moment dat; sistemele de operare moderne, privesc un proces ca fiind format din unul sau mai mute fire de execuție.

Execuția proceselor este un mecanism combinat hard şi soft. Componenta software care este implicată în administrarea proceselor este sistemul de operare, ale cărui sarcini sunt: crearea şi distrugerea proceselor şi firelor de execuție; planificarea proceselor; sincronizarea proceselor; comunicarea între procese; manipularea interblocării proceselor.

M1.U1.10. Generarea, configurarea şi lansarea în execuție a sistemului de operare

În cazul generațiilor de calculatoare mai vechi, sistemul de operare este destinat numai unui anumit tip de calculator.

Exemplu. Sistemul SIRIS este destinat familiei de calculatoare FELIX C-256,512,1024, iar sistemul de operare RSX este destinat minicalculatoatelor compatibile PDP, printre care se numără şi cele româneşti CORAL si IDEPENDENT.

Sistemele de operare moderne sunt realizate pentru a lucra pe o clasă de sisteme de calcul, care în multe cazuri diferă prin configurația lor. Sistemul trebuie să fie generat(sau configurat) pentru fiecare sistem de calcul în parte, proces care se numeşte **generarea sistemului**(SYSGEN). Programul SYSGEN citeşte dintr-un fişier sau cere informațiile care se referă la configurația hardware a sistemului respectiv. Prin acest proces sunt determinate următoarele informații:

►Ce fel de CPU este folosită? Care dintre opțiunile(set de instrucțiuni cod maşină extins, instrucțiuni aritmetice în virgulă flotantă etc) sunt instalate? Dacă sistemul este multi-procesor, fiecare CPU va fi descrisă.

►Câtă memorie este disponibilă?

►Care unități sunt disponibile? Sistemul trebuie să cunoască adresele acestor unități, adresele întreruperilor corespunzătoare unităților, precum şi nişte caracteristici tehnice ale unităților.

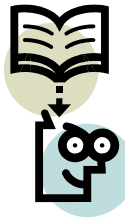
►Setarea unor parametri ai sistemului de operare, cum ar fi de exemplu, numărul şi dimensiunea zonelor tampon utilizate, numărul maxim de procese care pot fi lansate la un moment dat, tipul algoritmului de planificare utilizat de CPU etc.
După stabilirea acestor informații, sistemul de operare poate fi utilizat în mai multe moduri:

- administratorul de sistem poate modifica codul sursă al sistemului de operare, situație în care toate programele sursă vor fi recompilate şi vor fi refăcute legăturile;
- selectarea unor module noi care fac parte dintr-o bibliotecă precompilată, fiind necesară numai refacerea legăturilor;
- selectarea unor opțiuni se face în timpul execuției, situație caracteristică sistemelor de operare moderne.

Unul dintre serviciile de bază ale unui **SO** este acela de a se putea **autoîncărca** de pe disc în memoria internă şi de a se autolansa în execuție. Această acțiune se desfăşoară la fiecare punere sub tensiune a unui **SC**, precum şi atunci când utilizatorul doreşte să reîncarce **SO,** fiind cunoscută sub numele **încărcare sau lansare în execuție** a **SO**. Pentru lansare se utilizează un mecanism combinat **hard** şi **soft**, numit **bootstrap**.

Mecanismul **bootstrap** intră în lucru la apăsarea butonului de pornire <START> şi cuprinde următoarele etape:
1. Se citeşte din memoria ROM un număr de locații consecutive. Aceste locații conțin instrucțiunile de copiere ale programului care va încărca nucleul SO. Motivul pentru care acest program este stocat în memoria ROM, este că aceasta nu trebuie să fie inițializată şi că procesul respectiv se află la o adresă fixă, de unde poate fi lansat în cazul punerii sub tensiune sau re-setării sistemului. De asemenea, deoarece ROM este o memorie "read-only", ea nu poate fi virusată.
2. Se lanseză în execuție programul citit la pasul anterior, care citeşte de pe un disc un program mai complex, care încarcă nucleul sistemului de operare.
3. Se lansează în execuție programul citit la pasul anterior care va încărca nucleul sistemului de operare.
Observație. Se pune firesc întrebarea, de ce programul care încarcă nucleul SO nu este citit din memoria ROM? Răspunsul este că instalarea unui alt sistem de operare, ar însemna modificarea acestui program, deci a conținutului memoriei ROM, care este o problemă destul de costisitoare.

Să ne reamintim...



Calculatoarele furnizează un mecanism prin care diferite componente (hard sau soft) întrerup evoluția normală a procesorului, pentru a semnala acestuia diverse evenimente apărute. Rolul principal al întreruperilor este de a îmbunătăți utilizarea procesorului.
Apelurile de sistem furnizează o interfață între un proces şi sistemul de operare. Prin intermediul acestora, un program utilizator comunică cu SO şi cere anumite servicii de la acesta.
În cadrul sistemelor de operare, procesul reprezintă o entitate activă, un program în execuție ale cărui instrucțiuni sunt parcurse secvențial şi executate de către unitatea centrală a calculatorului.
Sistemele de operare moderne sunt realizate pentru a lucra pe o clasă de sisteme de calcul, care în multe cazuri diferă prin configurația lor. Sistemul trebuie să fie generat(sau configurat) pentru fiecare sistem de calcul în parte, proces care se numeşte **generarea sistemului**.

Înlocuiți zona punctată cu termenii corespunzători.

1. Când se lansează execuția unei operații de I/O, ... nu trebuie să aştepte terminarea ei; în acest timp, el poate servi un alt .....şi i se va transmite când s-a terminat de efectuat operația de ..... printr-o .....
2. Apelurile de sistem furnizează o .....între un proces şi .....
3 .În cadrul sistemelor de operare, procesul reprezintă o ....., un program în .....ale cărui instrucțiuni sunt parcurse ..... şi executate de către .....
4. Unul dintre serviciile de bază ale unui ..... este acela de a se putea ..... de pe disc în ..... şi de a se ..... în execuție.

M1.U1.11. Test de evaluare a cunoştințelor

Marcați varianta corectă.

1. Monitorul rezident:

a) Realizează prelucrarea serială a job-urilor. ☐
b) Permite alocarea memoriei cu partiții fixe. ☐
c) Lucrează în regim de multiprogramare. ☐
d) Execută operațiile de I/O în paralele cu efectuarea unor calcule. ☐

2. Conceptul de "spooling" a apărut odată cu:

a) Apariția hard-discurilor. ☐
b) Apariția generației a II-a de calculatoare. ☐
c) Apariția benzilor magnetice. ☐
d) Apariția cartelelor perforate. ☐

3. Controller-ul de unitate este:

a) O componentă software. ☐
b) O componentă a CPU. ☐
c) O interfață hardware a unității de I/O respective. ☐
d) Un registru al unității centrale. ☐

4. Nucleul SO este:

a) Componenta SO utilizată în execuția proceselor. ☐
b) Componenta SO executată în mod supervizor. ☐
c) Componenta SO utilizată în administrarea unităților. ☐
d) Componenta SO executată în mod utilizator. ☐

5. Un proces este:

a) Un program scris într-un limbaj de programare. ☐
b) Un fişier executabil. ☐
c) Un program în execuție. ☐
d) Un "buffer". ☐

6. Care dintre componentele sistemului de operare sunt scrise în limbajul de asamblare al calculatorului gazdă:

a) Nucleul ☐
b) Bibliootecile ☐
c) Driverele de memorie. ☐
d) Driverele de dispozitiv ☐

7. Instrucțiunile care pot fi executate numai în modul supervizor se numesc:

a) Instrucțiuni cod maşină. ☐
b) Instrucțiuni de asamblare. ☐
c) Instrucțiuni privilegiate. ☐
d) Instrucțiuni de comutare. ☐

8. Rolul principal al întreruperilor este de:

a) A îmbunătății utilizarea procesorului. ☐
b) A îmbunătății utilizarea nucleului. ☐
c) A îmbunătății utilizarea sistemului de operare. ☐
d) A îmbunătății utilizarea sistemului de fişiere. ☐

9. Apelurile de sistem furnizează:

a) O interfață între un proces şi controllerul de unitate. ☐

b) O interfață între un proces şi sistemul de operare. ☐

c) O interfață între un proces şi CPU. ☐

d) O interfață între un proces şi un disc ☐

10. Programul care încarcă nucleul sistemului de operare se citeşte:

a) Din memoria RAM. ☐

b) De la tastatură. ☐

c) Din memoria ROM. ☐

d) De pe un disc. ☐

M1.U1.12. Rezumat. Evoluția sistemelor de calcul, reflectată în cele patru generații de calculatoare existente până în prezent, este strâns legată şi de evoluția componentelor software, în care sistemul de operare ocupă un loc central. Dacă primele calculatoare electronice apărute nu au beneficiat de un sistem de operare, odată cu generația a doua de calculatoare apare conceptul de **monitor rezident**. Acesta era o formă rudimentară de sistem de operare, care a permis prelucrarea serială a lucrărilor şi automatizarea înlănțuirii fazelor unei lucrări.

Multiprogramarea, apărută odată cu generația a treia de calculatoare, a permis utilizarea mult mai eficientă a unității centrale a unui sistem de calcul. Apar conceptele de **driver** de unitate şi **întrerupere,** care permit o manipulare mai eficientă şi facilă a unităților de I/O. De asemenea, apar conceptul de **spooling**, care permite o utilizare mai eficientă a resurselor unui SC. Odată cu generația a patra de calculatoare, apar noi concepte cum ar fi **time-sharing**, **redirectarea** şi **legarea în pipe**, care urmăresc interactivitatea sistemelor şi un timp de răspuns cât mai mic. Conceptul de lucarare(job) este înlocuit cu cel de proces(task) şi apare conceptul de **multi-tasking**.

Prezentarea arhitecturii Von Neumann a unui sistem de calcul, care a rămas un model viabil şi pentru calculatoarele din zilele noastre este utilă în înțelegerea modului de interacțiune dintre sistemul de operare şi componentele hardware ale sistemului de calcul. Pe baza acestui model, se poate face o diferențiere a funcțiilor şi componentelor unui sistem de operare. Astfel, în ceea ce priveşte execuția proceselor, se prezintă pe scurt ce face CPU şi care este rolul sistemului de operare în gestiunea proceselor, aspect care va fi detaliat într-un capitol special. De asemenea, legat de administrarea unităților, se scoate în evidență rolul controllerelor de unitate, ca şi componente hard precum şi al celor de interfață software. De asemenea, tot legat de executarea unor operații de I/O, se prezintă rolul nucleului unui sistem de operare.

Autoîncărcarea, generarea şi configurarea sistemelor de operare reprezintă concepte care au fost dezvoltate în timp şi prin care este optimizată activitatea sistemului de operare.

Modulul 2. Gestiunea proceselor

Cuprins

Introducere

Procesul este abstractizarea unui program în execuție şi reprezintă elementul computațional de bază în sistemele de calcul moderne. Toate sistemele de operare moderne au fost construite în jurul conceptului de proces. Multe probleme pe care sistemul de operare trebuie sa le rezolve, sunt legate de procese.

Administrarea proceselor se referă la toate serviciile oferite de sistemul de operare pentru gestionarea activității tuturor proceselor care sunt executate la un moment dat. Pentru a realiza această activitate, sistemul de operare administrează mai mute structuri de date care vor fi detaliate în cadrul modulului. După definirea conceptului de proces, sunt prezentate stările în care acesta se poate afla la un moment dat, precum şi tranzițiile de la o stare la alta. Conceptul de fir de execuție, constituie o perfecționare a modelului de proces tradițional şi pe care se bazează sistemele de operare moderne.

În condițiile multiprogramării, sistemul de operare trebuie să realizeze comutarea CPU de la un proces la altul, în vederea creşterii gradului de utilizare al acesteia şi servirii echitabile a tuturor proceselor aflate în execuție la un moment dat. Servirea de către CPU a proceselor se realizează pe baza unor metode de planificare a execuției acestora, prin care se urmăreşte atât o servire echitabilă a proceselor, cât şi servirea prioritară a unora dintre ele, cum este cazul celor care aparțin sistemului de operare.

De asemenea, alocarea resurselor sistemului de calcul către procese, se face de către sistemul de operare, pe baza unei anumite strategii. Este discutată problema interblocării, care poate apare la alocarea de resurse proceselor. Sunt abordate condițiile în care apare interblocarea, strategiile folosite în abordarea acestei probleme, precum şi legătura dintre interblocarea şi sincronizarea proceselor.

Competențe

La sfârşitul acestui modul studenții vor fi capabili să:

- să înțeleagă conceptul general de proces;
- să înțeleagă importanța utilizării firelor de execuție;
- să identifice stările unui proces;
- să explice metodele utilizate în planificarea execuției proceselor;
- să identifice situațiile în care poate apare interblocarea proceselor;
- să explice metodele utilizate pentru înlăturarea stării de interblocare
- să implementeze algoritmii prezentați într-un limbaj de programare cunoscut.

Unitatea de învățare M2.U1. Coordonarea execuției proceselor

Cuprins

M2.U1.1. Introducere

Termenul de proces, poate fi definit prin una din următoarele modalități:
- Un program în executie.
- O instanță a unui program ce se execută pe un calculator.
- O entitate ce poate fi asignată şi executată de un processor.
- O unitate a unei activități caracterizată de execuția unei secvențe de instructiuni, o stare curentă şi o mulțime de resurse ale sistemului asociată.

Elementele esentiale ale unui proces sunt **codul programului** (care poate fi partajat cu un alt procese care execută acelaşi program) şi un **set de date** asociat acelui cod.

În condițiile multiprogramării, procesele trec dintr-o stare în alta, în funcție de cererile formulate de fiecare proces şi resursele sistemului de calcul disponibile la un moment dat. În momentul trecerii dintr-o stare în alta, când procesul respectiv trebuie să cedeze CPU către alt proces se produce comutarea de context, care este o operație costisitoare din punctul de vedere al procesorului. Pentru a se evita această situație, s-a dezvoltat conceptul de fir de execuție; acesta corespunde unei activități din cadrul programului respectiv şi comutarea CPU de la un fir la altul este mult mai puțin costisitoare.

În contextul multiprogramării, procesele care se executa în cadrul unui sistem de cacul sunt cooperante; apare astfel problema sincronizării execuției proceselor, în comunicarea între ele şi accesarea anumitor resurse necesare.

Optimizarea activității CPU precum şi apariția sistemelor de calcul multiprocesor, impune utilizarea conceptului de fir de execcuție, care corespunde unei activități executate în cadrul unei aplicații.

M2.U1.2. Obiectivele unității de învățare

Această unitate de învățare îşi propune ca obiectiv principal o introducere a studenților în problematica gestiunii proceselor. La sfârşitul acestei unități de învățare studenții vor fi capabili să:

- înțeleagă şi să explice tranzițiile unui proces dintr-o stare în alta;
- înțeleagă şi să explice modalitățile de creare a proceselor;
- înțeleagă şi să explice cauzele terminării proceselor;
- înțeleagă şi să explice metodele de coordonare a execuției proceselor;
- înțeleagă şi să explice utilizarea firelor de execuție.

Durata medie de parcurgere a unității de învățare este de 3 ore.

M2.U1.3. Stările unui proces.

Starea unui proces este definită ca mulțimea activităților executate de acesta. În cele ce urmează vom presupune că avem un sistem de calcul cu un singur procesor(CPU). Stările unui proces sunt:
New: corespunde definirii procesului. Procesul primeşte un identificator, se crează blocul de control al procesului (despre care vom discuta imediat), se alocă şi se construiesc tabelele necesare execuției procesului. Pentru o bună funcționare a sistemului, se pune problema limitării numărului de procese; pentru procesul respectiv se crează structurile necesare, urmând ca execuția să se facă în momentul în care sistemul poate aloca resursele necesare.
Run: procesul este în execuție, adică instrucțiunile sale sunt executate de CPU;
Wait: procesul aşteaptă apariția unui anumit eveniment, cum ar fi terminarea unei operații de I/O sau primirea unui semnal;
Ready: procesul este gata de execuție aşteptînd să fie servit de către procesor;
Finish: terminarea execuției procesului.

În sistemele cu multiprogramare, procesele trec dintr-o stare în alta, în funcție de specificul fiecăruia sau de strategia de planificare adoptată. În figura 2.1.1 este prezentată diagrama tranzițiilor unui proces în timpul execuției.

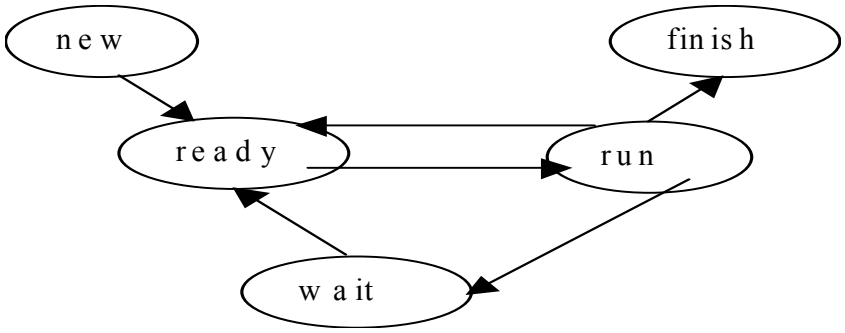


Figura 2.1.1 Diagrama stărilor şi tranzițiilor unui proces

Tranzițiile posibile sunt:
null→new: Un proces nou este creat pentru a executa un program.
new→ready: Înseamnă că procesul este luat în considerare pentru a fi executat; se realizează când sistemul dispune de resursele necesare pentru a executa un nou proces şi numărul proceselor din sistem nu depăşeşte o anumită valoare.
ready→run: Se produce atunci când procesului îi este alocat un procesor.
run→ready: Se produce atunci când, conform unei politici de planificare, procesorul trebuie alocat unui alt proces.
run→wait: Se produce atunci când procesorul întâlneşte o cerere de executare a unei operații de intrare/ieşire.
wait→ready: Se produce atunci când operația de I/O s-a terminat.
run→finish: Înseamnă terminarea execuției procesului.

M2.U1.4. Crearea şi terminarea proceselor

Următoarele evenimente conduc la crearea unui nou proces:
- primirea spre execuție a unui nou job (în cazul sistemelor cu prelucrare în loturi);
- loginarea unui nou utilizator (în cazul sistemelor interactive);

- crearea procesului de către sistemul de operare pentru a oferi un serviciu unui proces existent în sistem.
- declararea în cadrul unui proces (proces tată) a unui nou proces (proces fiu), care se vor executa în paralel.

Când un nou proces este adăugat celor în curs de execuție, sistemul de operare construiește o structură de date necesară administrării lui. Această structură de date se numește **blocul de control al procesului**(PCB – **P**rocess **C**ontrol **Bl**ock); este este o zonă de memorie, ce conține următoarele informții:
- **identificatorul procesului**, prin care se face distincția față de alte procese din sistem.
- **nivelul de prioritate** al procesului, în raport cu alte procese din sistem.
- **starea procesului**, ale cărei valori le vom prezenta imediat.
- valoarea **contorului de program**, care indică adresa următoarei instrucțiuni care urmează să fie executată pentru acest proces.
- valorile **regiştrilor unității centrale**; când apare o întrerupere şi CPU trece la execuția altui proces, valorile acestor regiştri şi ale contorului de program trebuie salvate, pentru a permite reluarea corectă a execuției procesului; când procesul primeşte din nou serviciile CPU, valorile salvate în PCB-ul procesului respectiv sunt atribuite regiştrilor acesteia.
- informații de **contabilizare**, referitoare la valoarea timpului CPU utilizat de proces, valori limită ale timpului CPU, între care este executat procesul respective, conturi utilizator, numere ale altor procese etc; informații de I/O (lista unităților de I/O alocate procesului, lista fişierelor deschise de proces etc.).
- informații legate de operațiile de I/O (cereri de operații de I/O în curs de execuție, perifericele asignate procesului etc.)
- informații de **planificare** a CPU(prioritatea procesului, pointer către coada de planificare a execuției procesului etc.).
- informații cu privire la modalitatea de **alocare** a memoriei interne;
- un pointer către un alt PCB.

Observații.
1. Toate PCB-urile asociate proceselor din sistem active la un moment dat formează o listă alocată dinamic.
2. Informațiile conținute în PCB-ul unui proces permite comutarea CPU de la un proces la altul, astfel încât reluarea execuției procesului să se facă cu instrucțiunea următoare ultimei instrucțiuni executate de CPU.

Terminarea execuției proceselor se poate realiza din următoarele motive:
- **Terminare normală**. Procesul execută o rutină a sistemului de operare prin care semnalează că a atins şi executat ultima instrucțiune cod maşină din fişierul executabil.
- **Depăşirea intervalului de timp** alocat de sistem.
- **Memorie nedisponibilă**. Procesul cere mai multă memorie decât cea pe care o poate oferi sistemul.
- **Violarea memoriei**. Procesul încearcă să acceseze o locație de memorie la care nu are permis accesul.
- **Eroare de protecție**. Procesul încearcă să acceseze o resursă într-un mod în care nu-i este permis.

Exemplu: scrierea într-un fişier asupra căruia nu are acest drept.

- **Eroare aritmetică**. Procesul încearcă să execute o operație interzisă .

Exemplu: împărţire la zero sau un calcul al cărui rezultat depăşeşte valoarea maximă stabilită prin hardware.

- **Depăşirea timpului de aşteptare** pentru producerea unui eveniment.
- **Eşec într-o operaţie de I/O**

Exemplu:imposibilitatea găsirii unui fişier, nerealizarea operaţiilor de citire sau scriere după un anumit număr de încercări, lansarea unei operaţii invalide-citirea de la imprimantă.

- **Instructiune invalidă**. Procesul încearcă să execute o instructiune inexistentă.
- **Instructiune privilegiată**. Procesul încearcă să execute o instructiune rezervată sistemului de operare.
- **Date eronate**. Tipul datelor referenţiate este eronat sau este invocată o variabilă neiniţializată.
- **Intervenţia** unui operator uman sau a sistemului de operare în situaţii în care execuţia procesului duce la blocarea sistemului de calcul.

Crearea proceselor presupune parcurgerea următorilor paşi:

1. Asignarea unui identificator unic procesului nou creat. În acest moment, se adaugă o nouă intrare la tabela proceselor active din sistem.

2. Alocarea spaţiului pentru proces. Aceasta include toate elementele necesare imaginii procesului. Sistemul de operare trebuie să cunoască dimensiunea spaţiului de memorie necesar execuţiei procesului (pentru încărcarea codului, pentru date şi stiva utilizator). Aceste valori pot fi asignate implicit de sistemul de operare pe baza tipului procesului care va fi creat sau a unor cerinţe formulate la crearea jobului. Dacă un proces este creat în contextul altui proces, procesul tată poate transmite valorile necesare sistemului de operare.

3. Iniţializarea blocului de control al procesului. Partea de identificare a procesului conţine identificatorul acestuia şi pe cel al procesului tată. Porţiunea de informaţii de stare utilizată de către procesor conţne mai multe intrări, dintre care o mare parte sunt iniţializate cu zero. Fac excepţie contorul de program (setat cu valoarea punctului de intrare in program) şi pointerii de stivă de sistem. Partea de informaţii de control a procesului este initializată pe baza unor valori standard implicite, la care se adaugă unele valori pentru attribute care au fost cerute pentru acest proces.

Exemplu: Starea procesului este iniţializată cu `Ready`. Prioritatea este setată implicit la valoarea minimă, dacă nu se specifică o valoare superioară. Iniţial, procesul nu posedă resurse (periferice, fişiere etc.), cu excepţia situaţiei în care face o cerere specială sau le moşteneşte de la procesul tată.

4. Setează legăturile potrivite.

Exemplu: Setarea legăturilor din coada de planificare a execuţiei proceselor, aflate în starea ready (administrată de sistemul de operare).

5. Crearea sau expandarea altor structuri de date.

Exemplu: Fişierul cu informaţii de contabilizare.

Să ne reamintim...

Starea unui proces este definită ca mulțimea activităților executate de acesta. Un proces se poate afla într-una dintre stările: new (corespunde definirii procesului), run (procesul este în execuție), wait (procesul așteaptă apariția unui anumit eveniment), ready (procesul este gata de execuție așteptînd să fie servit de către procesor), finish (terminarea execuției procesului). În sistemele cu multiprogramare, procesele trec dintr-o stare în alta, în funcție de specificul fiecăruia sau de strategia de planificare adoptată.

Următoarele evenimente conduc la crearea unui nou proces: primirea spre execuție a unui nou job (în cazul sistemelor cu prelucrare în loturi); loginarea unui nou utilizator (în cazul sistemelor interactive); crearea procesului de către sistemul de operare pentru a oferi un serviciu unui proces existent în sistem; declararea în cadrul unui proces (proces tată) a unui nou proces (proces fiu), care se vor executa în paralel.

Când un nou proces este adăugat celor în curs de execuție, sistemul de operare construieşte o structură de date necesară administrării lui. Această structură de date se numeşte **blocul de control al procesului.**

Terminarea execuției proceselor se poate realiza din următoarele motive:terminare normală, depăşirea intervalului de timp alocat de sistem, memorie nedisponibilă, violarea memoriei, eroare de protecție, eroare aritmetică, depăşirea timpului de aşteptare, eşec într-o operație de I/O, instructiune invalidă, instructiune privilegiată, date eronate, intervenția unui operator uman sau a sistemului de operare.

Înlocuiți zona punctată cu termenii corespunzători.

1. În sistemele cu multiprogramare, procesele trec dintr-o ..... în alta, în funcție de ..... sau de ..... adoptată.
2. Când un nou ..... este adăugat celor în curs de ....., sistemul de operare construieşte o ..... necesară administrării lui. Această structură de date se numeşte .....
3. Informațiile conținute în PCB-ul unui proces permite ..... de la un proces la altul, astfel încât reluarea execuției ..... să se facă cu instrucțiunea următoare ultimei instrucțiuni executate de CPU.
4. Terminarea normală a procesului presupune că acesta execută o ..... prin care semnalează că a atins şi executat ultima ..... din fişierul .....

M2.U1.5 Sincronizarea proceselor

Despre un proces se spune că este **independent** dacă execuția lui nu afectează sau nu poate fi afectată de execuția altor procese din sistem. Un astfel de proces poate fi **oprit sau repornit** fără a genera efecte nedorite, este **determinist**(ieşirile depind numai de starea de intrare), **reproductibil** (rezultatele sunt totdeauna aceleaşi, pentru aceleaşi condiții de intrare), **nu partajează date** cu alte procese din sistem. Dacă execuția unui proces poate fi afectată de execuția altor procese din sistem sau poate influența stările unor procese, atunci spunem că procesul este **cooperant**. În acest caz, procesul partajează date împreună cu alte procese din sistem, evoluția lui nu este deterministă, fiind influențată de stările acestora, nu este reproductibil etc. **Sincronizarea** proceselor reprezintă un mecanism prin care un proces activ este capabil să blocheze execuția altui proce (trecerea din starea run în ready şi invers), sau să se autoblocheze(să treacă în starea wait), sau să activeze un alt proces.

Problema secțiunii critice. Considerăm un sistem în care există n procese cooperante p_0, p_1,.., p_{n-1}. De exemplu, să presupunem că două dintre procese, p_0 şi p_1 au acces la o variabilă v, primul proces scade o valoare c_0, iar al doilea adaugă o valoare c_1 la v. Dacă secvența de operații executate de cele două procese asupra variabilei v se derulează necontrolat, atunci rezultatele sunt total imprevizibile. Putem extinde problema enunțată asupra fişierelor, bazelor de date, tabele din memorie etc. O astfel de resursă logică partajată de către două procese se numeşte **resursă critică**. Partea de program(segmentul de cod) în care un proces accesează o resursă critică se numeşte **secțiune critică**. Problema care se pune este stabilirea unor reguli după care un proces poate să intre în propria sa secțiune critică şi să comunice celorlate procese cooperante când a părăsit-o. Astfel, structura unui proces este:

```
do{<secțiune de intrare>
   <secțiune critică>
   <secțiune de ieşire>
   <secțiune rămasă>
}while(1);
```

O soluție corectă a problemei secțiunii critice trebuie să satisfacă următoarele condiții:

- **excludere mutuală**: la un anumit moment, un singur proces îşi execută propria lui secțiune critică;
- **evoluție(progres)**: un proces care nu este în secțiunea sa critică, nu poate să blocheze intrarea altor procese în propriile lor secțiuni critice, atunci când acestea doresc acest lucru;
- **aşteptare limitată**: între momentul formulării unei cereri de acces în propria secțiune critică de către un proces şi momentul obținerii accesului, trebuie acordat un număr limitat de accese celorlalte procese în propriile lor secțiuni critice.

Semafoare. Conceptul de semafor a fost introdus de Dijkstra. Un semafor s este o pereche (v,q); v este un întreg ce reprezintă valoarea semaforului, fiind inițializat la crearea semaforului cu o valoare v_0 iar q este o coadă, inițial vidă în care sunt introduse procesele care nu reuşesc să treacă de semaforul s.

Asupra semaforului pot acționa două operații indivizibile, W(wait) şi S(signal), executate asupra unui anumit proces. Operația W poate fi considerată ca o încercare de trecere a semaforului iar operația S ca o permisiune de trecere. Efectul celor două primitive este descris în continuare, în condițiile în care p este un proces care încearcă să treacă de semaforul s.

a) Efectul primitivei W

```
do {v--:
    If (v<0) // procesul p  se introduce în coadă
         {
              Stare(p):= wait;
              q← p}
         } while(1):
```

b) Efectul primitivei S

```
do { v++:
 If  (v <= 0)  // procesul p este scos din coadă şi activat
      {    q→ p;
           Stare(p): = ready}
           } while(1):
```

Observații. 1. Dacă nw şi ns reprezintă numărul primitivelor W, respectiv S executate pe semaforul s, atunci

$$v = v_0 - nw + ns \qquad (2.1.1)$$

2. Valoarea inițială v_0 este un întreg pozitiv.
3. La un moment dat, dacă valoarea semaforului v este negativă, respectiv pozitivă, valoarea ei absolută reprezintă numărul proceselor blocate în coadă, respectiv numărul proceselor care pot trece de semaforul s.
4. Dacă presupunem că la un moment dat numărul proceselor care au trecut cu succes de semaforul s este nt, atunci evident că $nt \geq nw$. De asemenea:

$$nt = \min\{v_0 + ns, nw\} \qquad (2.1.2)$$

5. Modificarea valorilor întregi ale semafoarelor trebuie să fie executate indivizibil, adică atunci când un process modifică valoarea semaforului nici un alt proces nu poate face acest lucru.

Utilizarea semafoarelor în gestionarea resurselor critice. Putem folosi semafoarele pentru a rezolva problema unei secțiuni critice, partajată de către `n` procese. Procesoarele folosesc în comun un semafor mutex (**mut**ual **ex**clusion), a cărui valoare inițială este 1. Fiecare proces p_i este organizat astfel:

```
do {
     W(mutex);
     <secțiune critică>
     S(mutex);
     <secțiune rămasă>
  } while(1);
```

Observații.
1. Secțiunea critică este precedată de o primitivă `W`, care aplicată semaforului `mutex` va permite intrarea în secțiunea critică numai în cazul în care resursa critică cerută este liberă. După execuția resursei critice, urmează execuția primitivei `S` asupra aceluiaşi semafor, al cărei efect este eliberarea resursei şi eventual deblocarea unui proces din coada semaforului, dacă aceasta este nevidă, care să treacă în secțiunea sa critică.
2. Operațiile `W` şi `S` trebuie să fie indivizibile.
3. Pentru a demonstra corectitutinea soluției, mai întâi vom proba proprietatea de excludere mutuală. Numărul proceselor care se se află la un moment dat în secțiunea critică este `nt-ns`. Deoarece valoarea inițială a semaforului este `1`, conform relației (2.1), avem inegalitatea $nt-ns \leq 1$, care are semnificația că cel mult un proces se poate afla la un moment dat în secțiunea sa critică. Mai trebuie să arătăm că dacă nici un proces nu se află în secțiunea sa critică, atunci intrarea unui proces în secțiunea sa critică va avea loc după un interval finit de timp, care este echivalent cu a arăta că dacă nici un proces nu se află în secțiunea sa critică, atunci nu există procese blocate în coada de aşteptare a semaforului. Dacă nici un proces nu se află în secțiunea sa critică, atunci `nt=ns`. Dacă ar exista procese blocate în coada semaforului `mutex`, atunci `nt>nw` şi deci în conformitate cu (2.2) `nt=1+ns`, ceea ce reprezintă o contradicție.

M2.U1.6. Probleme clasice de coordonare a proceselor

Problema producător/consumator. Procesele de tip producător/ consumator apar destul de frecvent în cadrul sistemelor de operare, de exemplu atunci când două procese comunică între ele. Procesul de tip producător **generează informațiile**(mesajele), care vor fi **folosite** de către procesul de tip consumator. Această problemă poate fi rezolvată prin utilizarea unei zone de memorie accesibilă tuturor proceselor care cooperează. Ea este structurată în funcție de natura informațiilor schimbate şi de regulile care guvernează comunicația propriu-zisă. Această zonă comună este o resursă critică, ce trebuie protejată.

Fie procesele p_i şi p_j care comunică între ele. Cele două procese partajează o zonă comună ZC, care este divizată în n celule-mesaj, în care se depun (producătorul), respectiv se extrag (consumatorul) mesaje de dimensiune fixă. Procesele p_i şi p_j au o evoluţie ciclică ce cuprinde operaţiile de producere şi de depunere, respectiv de extragere şi de consumare a unui mesaj din ZC, aşa cum este redat în figura 2.1.2. Depunerea, respectiv extragerea unui mesaj din zona comună, trebuie să satisfacă următoarele restricţii:

a) Consumatorul nu poate să extragă un mesaj, pe care producătorul este în curs să-l depună, adică operaţiile de depunere şi extragere trebuie să se execute în excludere mutuală la nivelul mesajului.
b) Producătorul nu poate să depună un mesaj atunci când ZC este plină, iar dacă ZC este vidă, consumatorul nu poate extrage nici un mesaj.

Aceste situaţii limită trebuie detectate şi semnalate proceselor care depind de ele. Condiţia a) poate fi satisfăcută prin introducerea unui semafor de excludere mutuală, asociat unei zone comune ZC, mutex(ZC), iniţializat cu valoarea 1. Restricţia b) se poate respecta, utilizând semafoarele s_0 şi s_1 pentru a înregistra, în orice moment, starea de ocupare a zonei ZC; s_0, iniţializat cu 0 va indica numărul de celule-mesaj depuse în ZC, iar s_1, iniţializat cu n, va indica numărul celulelor-mesaj libere în ZC.

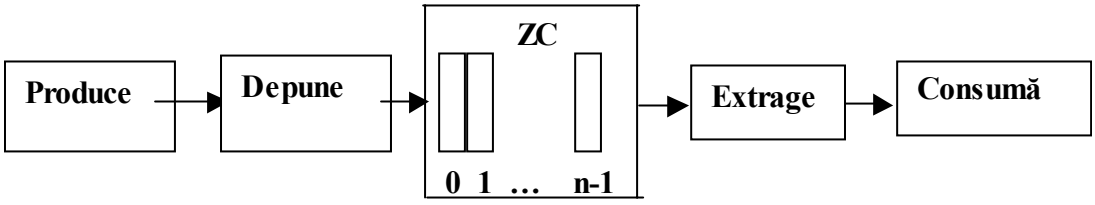


Figura 2.1.2 Structura zonei comune ZC

Descrierea proceselor producător/consumator p_i, respectiv p_j este prezentată în continuare.

```
{pi: proces producător de mesaje}
do {
      construieşte mesaj:
      W(s1);
      W(mutex(ZC));
      ZC←mesaj;
      S(mutex(ZC));
      S(s1);
      } while(1);

{pj: proces consumator de mesaje}
do {
      W(s0);
      W(mutex(ZC));
      ZC→mesaj;
      S(mutex(ZC));
      S(s0);
      Consumă mesaj
      } while(1);
```

Problema cititori/scriitori. Atunci când o structură de date, de exemplu un fişier sau o bază de date este accesată în comun de către mai multe procese concurente, dintre care unele

doresc doar **să citească** informații, iar altele doresc **să actualizeze** (scrie sau modifice) structura respectivă, cele două tipuri de procese se împart generic în **cititori** şi respectiv **scriitori.**

Se impun următoarele restricții:

a) Doi cititori pot accesa simultan obiectul partajat;
b) Un cititor şi un scriitor nu pot accesa în acelaşi timp obiectul disputat.

Pentru a respecta cele două condiții, accesul la resursă va fi protejat; problema de sincronizare astfel apărută se numeşte **cititori/scriitori.** Problema cititori/scriitori poate fi formulată în mai multe variante, toate implicând lucrul cu priorități. Cea mai simplă dintre acestea, impune ca nici un cititor să nu fie obligat să aştepte decât în cazul în care un scriitor a obținut deja permisiunea de utilizare a obiectului disputat. O altă variantă cere ca, de îndată ce un scriitor este gata de execuție, el să îşi realizeze scrierea cât mai repede posibil, adică dacă un scriitor aşteaptă pentru a obține accesul la obiectul disputat, nici un cititor nu va mai primi permisiunea de a accesa la resursă.

O soluție a primei variante a problemei cititori/scriitori permite proceselor să partajeze semafoarele `A`("Acces"), `Sc`("Scrie") şi variabila cu valori întregi `C`("contor"). `Sc`, inițializat cu `1` este folosit de ambele tipuri de procese. El asigură excluderea mutuală a scriitorilor, este folosit de către primul/ultimul cititor care intră/iese din propria secțiune critică, dar nu şi de către cititorii care intră/ies în/din aceasta în timp ce alți cititori se află în propriile secțiuni critice. Semaforul `A`, inițializat cu `1` este folosit pentru protejarea variabilei `C`, care este inițializată cu `0` şi are rolul de a memora numărul proceselor care citesc din obiectul partajat la momentul considerat. Primul cititor care accesează resursa critică(`C=1`) blochează accesul scriitorilor, iar ultimul cititor (`C=0`), deblochează accesul la obiectul partajat.

Descrierea celor două tipuri de procese este prezentată în continuare.

```
do {
        W(A);
        C++;
        if (C ==1 )
         W(Sc);
        S(A);
        <citirea>
        W(A); C--;
        if (C == 0) S(Sc);
}  while(1);
```

a) Procesul cititor

```
do
{
      W(Sc);
        <scrierea>
          S(Sc);
 } while(1);
```

b) Procesul scriitor

Să ne reamintim...

Dacă execuția unui proces poate fi afectată de execuția altor procese din sistem sau poate influența stările unor procese, atunci spunem că procesul este **cooperant**..

Sincronizarea proceselor reprezintă un mecanism prin care un proces activ este capabil să blocheze execuția altui proce sau să se autoblocheze sau să activeze un alt proces.

O resursă logică partajată de către două sau mai multe procese se numeşte **resursă critică**. Partea de program(segmentul de cod) în care un proces accesează o resursă critică se numeşte **secțiune critică**.

Problema producător/consumator se referă la partajarea unei zone de memorie. Problema cititori/scriitori se referă la partajarea unui fişier.

Înlocuiți zona punctată cu termenii corespunzători.

1. Sincronizarea proceselor reprezintă un mecanism prin care un proces ..... este capabil să ..... execuția altui proces sau să se ..... sau să activeze .....
2. Partea de program(segmentul de cod) în care un proces accesează o ..... critică se numeşte ..... critică.
3. Excluderea mutuală presupune că la un anumit ....., un singur ..... îşi execută .....
4. Un semafor s este o pereche (v,q); v este un întreg ce reprezintă ....., fiind inițializat la ..... semaforului cu o valoare v_0 iar q este o coadă, inițial ..... în care sunt introduse ..... care nu reuşesc să treacă de semaforul s.
5. Secțiunea critică este precedată de ....., care aplicată ..... va permite intrarea în ..... numai în cazul în care resursa ..... cerută este .....
6. Consumatorul nu poate să ....., pe care producătorul este ....., adică operațiile de depunere şi extragere trebuie să se execute în ..... la nivelul mesajului.

M2.U1.7. Fire de execuție

În mod tradițional, fiecărei aplicații îi corespunde un proces. Pe calculatoarele moderne se execută aplicații care, din punct de vedere logic, sunt structurate pe activități, care se pot desfăşura simultan.

Exemplu: În cadrul unui „browser" de WWW, o activitate poate fi afişarea unor texte sau imagini, pe când alta poate fi regăsirea unor informații de pe anumite servere dintr-o rețea de calculatoare. Un procesor de texte poate executa simultan afişarea de imagini sau texte, citirea unor comenzi de la tastatură şi verificarea greşelilor gramaticale sau de sintaxă ale textului introdus.

Modelul proceselor, se bazează pe două considerente principale:
- procesul dispune de o colecție de resurse, între care un **spațiu de memorie**, necesare execuției sale;
- procesul are un **singur fir de control**, adică la un moment dat cel mult o singură instrucțiune a procesului poate fi în execuție pe un procesor.

Într-un sistem de operare multiproces, una dintre operațiile de bază ale sistemului de operare, o constituie **comutarea proceselor,** conform unui algoritm de planificare. Comutarea proceselor implică salvarea de către nucleu a unor informații suficiente, care ulterior să fie restaurate, atunci când procesorul respectiv va primi din nou serviciile CPU, pentru ca execuția să fie reluată cu instrucțiunea următoare celei care a fost executată ultima. Aceste informații se referă la diverse categorii de resurse deținute de proces, ceea ce face ca operația de comutare a proceselor să fie relativ costisitoare din punctul de vedere al timpului procesor consumat pentru efectuarea ei. De asemenea, crearea unui nou proces în sistem este o operație de o complexitate deosebită.

Atât cerințele unor noi domenii de aplicații, cât şi dezvoltările tehnologice şi arhitecturale ale sistemelor de calcul(în special sistemele multiprocesor), au impus renunțarea la condiția de unicitate a firului de control în cadrul unui proces. A apărut astfel conceptul de **fir de execuție** (**thread**); mai multe asemenea fire pot coexista într-un proces, utilizând în comun resursele procesului. Avantajele principale fiind:
- **eliminarea costului** creării de noi procese;

• **simplificarea comutării**, pentru că numărul de informații salvate/restaurate va fi mai redus;
• posibilitatea ca o **aplicație interactivă să-şi continue execuția,** chiar dacă o parte a ei este blocată sau execută o operație mai lungă în timp; de exemplu, un browser de WWW menține interacțiunea cu utilizatorul, chiar dacă a început încărcarea unei imagini;
• **utilizarea arhitecturilor multiprocesor**, care permit ca fiecărui fir de control să-i fie alocat un procesor.

Firele de execuție pot coexista într-un proces(în acelaşi spațiu de memorie), evident cu necesitatea de a fi sincronizate atunci când solicită acces la resurse partajate, ca şi în cazul proceselor. Resursele proprii fiecărui fir de execuție sunt, în general limitate la: stivă, variabile locale şi numărător(contor) de program. Variabilele globale ale unui program vor fi implicit accesibile tuturor firelor din acel program.

Firele de execuție se împart în:
• **fire ale utilizatorului,** implementate de către o bibliotecă ce poate fi accesată de către utilizator şi care oferă posibilități de creare, planificare şi administrare a **thred**-urilor, fără a fi necesar suportul nucleului; acestora li se asigură spațiu de memorie în vederea execuției lor în spațiul utilizator;
• **fire ale nucleului sistemului de operare,** implementate de către sistemul de operare, care realizează crearea, planificarea şi administrarea lor în spațiul nucleului, ele realizând anumite operații protejate cerute de către procese.

Utilizarea firelor de execuție se poate realiza folosind mai multe modele:
• **mai multe fire utilizator care corespund la un fir al nucleului,** este caracterizat de eficiență, deoarece administrarea lor este realizată în spațiul utilizatorului, dar în cazul în care unul dintre firele utilizator generează un apele de sistem, prin care intră în starea de blocare, întregul proces va fi blocat; de asemenea, deoarece numai un fir de execuție poate, la un moment dat să acceseze nucleul, nu este posibil ca mai multe fire de execuție să se execute în paralel, pe un sistem multiprocesor.
• **un fir utilizator care corespunde la un fir al nucleului** permite concurența execuției, deoarece atunci când un fir de execuție a inițiat un apel de sistem care generează blocarea, celelalte fire de execuție se pot executa în continuare; dezavantajul constă în numărul mare de fire nucleu care trebuie create.
• **mai multe fire utilizator care corespund la mai multe fire ale nucleului(**în număr mai mic sau egal decât cele ale utilizatorului) este o situație specifică sistemelor multiprocesor, deoarece este posibil, ca la un moment dat, mai multe fire ale unei aplicații, care sunt servite de procesoare diferite, să lanseze un acelaşi apel de sistem, care să fie rezolvat de fire nucleu diferite.

Există mai multe variante conceptuale şi de implementare, după cum firele de execuție se manifestă numai în spațiul utilizator sau au reprezentare şi în nucleu. Un astfel de model este prezentat în figura 2.1.3.

Procesele uşoare (**lightweight process**, prescurtat LWP) pot fi considerate ca o corespondență între firele la nivelul utilizatorului şi firele din nucleu. Fiecare proces uşor este legat, pe de o parte de una sau mai multe fire utilizator, iar pe de altă parte, de un fir nucleu. Planificarea pentru execuție a LWP se face independent, astfel că LWP separate pot fi executate simultan în sisteme multiprocesor.

Figura 2.1.3 ilustrează diversele relații care se pot stabili între cele 4 tipuri de entități. Se observă, în primul rând că un LWP este vizibil şi în spațiul utilizator, ceea ce înseamnă că vor exista structuri de date corespunzătoare LWP atât în cadrul unui proces, cât şi în cadrul nucleului. Pe lângă situațiile ilustrate în figură, unde există numai fire de execuție nucleu asociate unor fire utilizator, nucleul poate conține şi fire care nu sunt asociate unor fire

utilizator. Asemenea fire sunt create, rulate şi apoi distruse de nucleu, pentru a executa anumite funcţii ale sistemului. Utilizarea de fire nucleu în locul proceselor, contribuie la reducerea costurilor de comutare în interiorul nucleului.

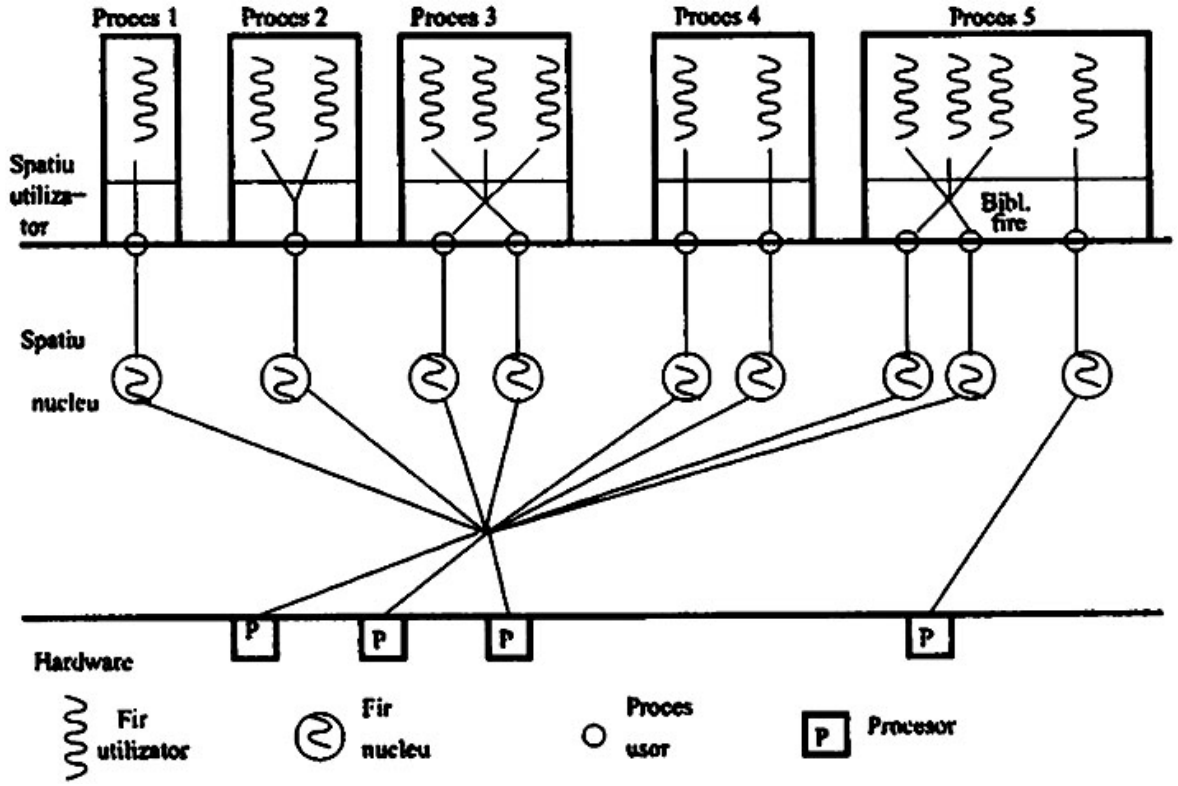


Figura 2.1.3. Modelul firelor de execuţie

Exemplu. Procesul 1 din figura 2.1.3 este un proces "clasic": el are un singur fir de execuţie la nivelul utilizatorului, legat cu un LWP. **Procesul 2** este o ilustrare a situaţiei în care firele de execuţie multiple sunt vizibile numai în spaţiul utilizator: toate sunt puse în corespondenţă cu un singur LWP, deci la un moment dat numai unul dintre firele procesului 2 poate fi în execuţie, pentru că planificarea se face la nivelul LWP. **Procesul 3** are mai multe fire de execuţie, multiplexate pe un număr mai mic de LWP. Aplicaţiile de acest gen conţin paralelism, al cărui grad poate fi controlat prin numărul de LWP puse la dispoziţie. **Procesul 4** are fiecare fir de execuţie pus în corespondenţă cu un LWP, ceea ce face ca nivelul de paralelism din aplicaţie să fie egal cu cel din nucleu. Acest aranjament este util dacă firele de execuţie ale aplicaţiei sunt frecvent suspendate cu blocare(fiecare fir suspendat cu blocare ţine blocat şi LWP-ul asociat). **Procesul 5** conţine atât fire care multiplexează un număr de LWP, cât şi un fir asociat permanent cu un LWP, care la rândul său este asociat permanent cu un procesor.

Să ne reamintim...
Un **fir de execuţie** (**thread**) corespunde unei activităţi din cadrul programului; mai multe asemenea fire pot coexista într-un proces, utilizând în comun resursele procesului. Avantajele utilizării firelor de execuţie sunt: **eliminarea costului** creării de noi procese, **simplificarea comutării**, posibilitatea ca o **aplicaţie interactivă să-şi continue execuţia,** chiar dacă o parte a ei este blocată sau execută o operaţie mai lungă în timp, **utilizarea arhitecturilor multiprocesor**, care permit ca fiecărui

fir de control să-i fie alocat un procesor.

Firele de execuție pot coexista într-un proces(în acelaşi spațiu de memorie), evident cu necesitatea de a fi sincronizate atunci când solicită acces la resurse partajate, ca şi în cazul proceselor. Firele de execuție se împart în **fire ale utilizatorului** şi **fire ale nucleului SO.** Prin intermediul proceselor uşoare firele utilizator accesează firele nucleului sistemului de operare.

Înlocuiți zona punctată cu termenii corespunzători.

1. Pe calculatoarele moderne se execută aplicații care, din punct de vedere ....., sunt structurate pe ....., care se pot desfăşura .....
2. Firele de execuție ale utilizatorului sunt implementate de către o ..... ce poate fi accesată de către ..... şi care oferă posibilități de ....., fără a fi necesar suportul .....
3. Fire a de execuție le nucleului sistemului de operare sunt implementate de către sistemul de operare, care realizează ..... lor în spațiul ....., realizând anumite operații ..... cerute de către .....
4. Procesele uşoare pot fi considerate ca o ..... între firele la nivelul .....şi firele din .....

M2.U1.8. Test de evaluare a cunoştințelor

Selectați varianta corectă.

1. Un proces blocat(aflat în starea **wait**) poate trece în starea:
   a) Run. ☐ c) Ready. ☐
   b) New. ☐ d) Finish. ☐
2. Structura de date care conține informații despre un proces se numeşte:
   a)Vectorul de control al procesului. ☐ c) Blocul de control al procesului. ☐
   b)Vectorul de control al sistemului ☐ d) Blocul de control al sistemului. ☐
3. În ce stare a unui proces se construieşte blocul său de control.
   a) Run. ☐ c) Ready. ☐
   b) New. ☐ d) Finish. ☐
4. Trecerea unui proces în starea de blocare se poate realiza din starea:
   a) Run. ☐ c) Ready. ☐
   b) New. ☐ d) Finish. ☐
5. Care dintre tranziții conduc la creşterea gradului de utilizare a CPU, în condițiile multiprogramării:
   a) run→ready ☐ c) run→finish ☐
   b) run→wait ☐ d) run→new. ☐
6. Condiția de evoluție (progres) însemnă că:
   a) Un proces care nu este în secțiunea sa critică, nu poate să blocheze încărcarea unui alt proces în memoria internă. ☐ c) Un proces care nu este în secțiunea sa critică, nu poate să blocheze accesul unui alt proces la imprimantă. ☐
   b) Un proces care nu este în secțiunea sa critică, nu poate să blocheze intrarea altor procese în propriile lor secțiuni critice, atunci când acestea doresc acest lucru. ☐ d) Un proces care nu este în secțiunea sa critică, nu poate să blocheze cititirea de date de la tastatură de către alt proces. ☐

7. Condiția de aşteptare limitată înseamnă:

a) Între momentul formulării unei cereri de acces în propria secțiune critică de către un proces şi momentul obținerii accesului, nu trebuie efectuate operații de intrare/ieşire. ☐

b) Între momentul formulării unei cereri de acces în propria secțiune critică de către un proces şi momentul obținerii accesului, trebuie acordat un număr limitat de accese celorlalte procese în propriile lor secțiuni critice. ☐

c) Între momentul formulării unei cereri de acces în propria secțiune critică de către un proces şi momentul obținerii accesului, nu trebuie încărcate în memorie alte procese. ☐

d) Între momentul formulării unei cereri de acces în propria secțiune critică de către un proces şi momentul obținerii accesului, nu trebuie acordat dreptul altor procese de a-şi accesa propriile resurse critice. ☐

8. Excluderea mutuală înseamnă că:

a) La un anumit moment, un singur proces îşi execută propria lui secțiune critică. ☐

b) La un anumit moment, un fişier poate fi accesat de un singur proces. ☐

c) La un anumit moment, un singur proces se află în starea `run`. ☐

d) La un anumit moment, un singur proces poate fi încărcat în memorie. ☐

9. Problema producător/consumator se referă la:

a) O metodă de comunicare între procese. ☐

b) O metodă de partajare a unui fişier. ☐

c) O metodă de gestionare a unei zone critice. ☐

d) O metodă de administrare a memoriei interne. ☐

10. Problema cititori/scriitori se referă la:

a)Partajarea unei resurse hardware. ☐

b)Partajarea unei zone de memorie. ☐

c) Partajarea unui fişier. ☐

d)Partajarea unui slot din memoria "cache". ☐

11. Firele de execuție pot coexista:

a) În cadrul aceluiaşi fişier. ☐

b) În cadrul aceluiaşi program. ☐

c). În cadrul aceluiaşi proces. ☐

d) În cadrul aceleiaşi pagini de memorie. ☐

12. Un fir de execuție corespunde:

a) Unei activități din cadrul aplicației respective. ☐

b) Unui bloc de control al unui proces. ☐

c). Unui fişier deschis în cadrul procesului respectiv. ☐

d) Unui periferic utilizat de către proces. ☐

13. Procesele uşoare pot fi considerate ca:

a) O corespondență între firele utilizatorului şi firele din nucleu ☐

b) O corespondență între între procesele lansate în execuție de un utilizator şi discurile utilizate. ☐

c)O corespondență între procesele lansate în execuție de un utilizator şi fişierele lui. ☐

d) O corespondență între firele şi procesele aceluiaşi utilizator. ☐

M2.U1.9. Rezumat

Procesele secvențiale reprezintă baza modelelor computaționale moderne; comportarea unui proces secvențial este definită de programul pe care procesele îl execută. Administratorul proceselor răspunde de crearea mediului în care este executat procesul, incluzând aici şi administrarea resurselor. La rândul lui, sistemul de operare se sprijină pe conceptul de proces. Toate procesele care formează sistemul de operare sunt lansate în execuție după ce a fost lansat procesul inițial (”boot”), prin care este încărcat sistemul de operare.

Pentru a ajunge fişier executabil, un program trece prin mai multe faze. Programul este scris în limbaj sursă. După faza de compilare, este transformat în modul obiect. Prin editarea legăturilor, se ajunge la fişierul executabil. Execuția proceselor este un mecanism combinat hard şi soft. Sistemul de operare este componenta soft care participă la execuția proceselor. Execuția procesului presupune şi alocarea unui spațiu de memorie. În acest sens, putem spune că administratorul proceselor şi cel al memoriei interne conlucrează în execuția proceselor.

În cursul execuției proceselor, acestea trec dintr-o stare în alta în funcție de cererile de resurse formulate, precum şi de disponibilitatea sistemului de a aloca resursele necesare la momentul respective.

În sistemele moderne, procesele nu lucrează independent, ele reprezintă entități care comunică între ele. Acestea oferă mecanisme prin care se poate crea o ierarhie de procese, ceea ce reprezintă o modalitate prin care un proces poate controla alte procese, iar comunicația între ele şi gestionarea unor resurse partajate se poate realiza mult mai eficient. În cursul execuției lor, procesele concurează la obținerea de resurse ale sistemului de calcul. Sistemul de operare realizează sincronizarea proceselor în accesarea acestor resurse.

Firele de execuție reprezintă un mecanism pe care se sprijină sistemele de calcul moderne, care oferă facilități perfecționate de execuție simultană a mai multor activități, în cadrul aceleaşi sarcini şi care permite o comutare mult mai eficientă a CPU, între elementele de calcul ce se execută la un moment dat.

Unitatea de învățare M2.U2. Planificarea execuției proceselor

Cuprins

M2.U2.1. Introducere

În condițiile multiprogramării, decizia de a aloca procesorul (sau unul dintre procesoare în cazul sistemelor multiprocesor) unui anumit proces din sistem este o decizie fundamentală care este luată cu sprijinul sistemului de operare. Această activitate urmăreşte îndeplinirea anumitor obiective ale sistemului: timp de răspuns cât mai mic; cresterea productivității sistemului, exprimată prin numărul de procese sau thread-uri executate pe unitatea de timp (oră); creşterea gradului de utilizare a CPU.

Planificariea execuției proceselor poate fi împărțită în:
• planificare pe termen lung - se referă la decizia de a adăuga un nou proces pentru a fi executat, pe lângă cele care se află în execuție (trecerea în starea new);
• planificare pe termen mediu - se referă la decizia de a adăuga un nou proces pentru a fi încărcat în memorie în vederea execuției (trecerea în starea ready);
• planificare pe termen scurt - se referă la decizia de a selecta unul dintre procesele aflate în starea ready pentru a primi serviciile CPU(trecerea în starea run).

În cadrul unității de învățare sunt prezentate diverse strategii de planificare, care au evoluat odată cu progresele realizate de către componentele software şi hardware. Dacă prelucrarea serială folosea strategii prin care un proces nu ceda CPU pe toată perioada execuției lui, strategiile utilizate de sistemele care se bazează pe partajarea timpului încearcă să răspundă cerințelor servirii echitabile a tuturor proceselor şi realizării unui timp de răspuns cât mai mic în cazul proceselor care au o prioritate înaltă.

În cele ce urmează, ne vom referi la metode de planificare pentru sistemele cu un singur procesor.

M2.U2.2. Obiectivele unității de învățare

Această unitate de învățare îşi propune ca obiectiv principal o inițiere a studenților în planificarea execuției proceselor. La sfârşitul acestei unități de învățare studenții vor fi capabili să:

- identifice componentele a planificatorului;
- înțeleagă şi să explice metodele de planificare a execuției proceslor;
- identifice implicațiile metodelor de planificare în creşterea performanțelor sistemelor de calcul;
- implementeze algoritmii prezentați.

Durata medie de parcurgere a unității de învățare este de 2 ore.

M2.U2.3. Concepte introductive

Planificare pe termen lung se referă la ce programe sunt admise de sistem pentru a fi prelucrate. Astfel, se controlează gradul de multiprogramare. Odată admis, un program sau job devine proces si el este adaugat la coada proceselor planificate pe termen scurt. În această situație sistemul trebuie să ia două decizii:

1. Când sistemul de operare poate să ia în considerare unul sau mai multe programe sau job-ri care să devină procese; o astfel de decizie ține cont de gradul de multiprogramare. Limitarea gradului de multiprogramare de către planificator urmăreşte să asigure proceselor în curs de execuție un timp rezonabil pe care să îl petreacă în sistem. De fiecare dată când se termină execuția unui proces, planificatorul decide dacă este cazul să creeze încă unul sau mai multe procese, dacă procentul de timp de lenevire al CPU depăşeşte un anumit prag.

2. Care dintre programe sau job-uri să devină procese. Această decizie se poate lua pe baza mai multor criterii:
- împărțirea pe clase de priorități a programelor;
- ce resurse cerute sunt disponibile la momentul respectiv.

Exemplu. Într-un sistem bazat pe partajarea timpului, atunci când un utilizator încearcă să se logineze, cererea poate să fie respinsă, deoarece cererea de execuție a procesului de loginare nu poate să fie satisfăcută din cauza suprasaturării sistemului, urmând ca cererea respectivă să fie luată în considerație după o nouă încercare de loginare a utilizatorului.

Planificarea pe termen mediu se poate realiza din diverse motive:
- trecerea din starea **new** în starea **ready;**
- revenirea din starea de blocare;
- reîncărcarea unui proces care a fost salvat pe disc(swaped); se întâlnea mai ales la sistemele care nu aveau memorie virtuală.

Planificarea pe termen scurt. Obiectivul principal al planificării pe termen scurt este de a aloca proceselor timpul CPU astfel încât să se optimize unul sau mai multe aspecte ale funcționării sistemului de calcul. În acest sens, sunt stabilite unul sau mai multe criterii, pe baza cărora se evaluează politicile de planificare. Aceste criterii pot fi orientate către utilizator sau către sistem.

Criteriile orientate către utilizator sunt:
• Timpul petrecut în sistem de proces - intervalul de timp dintre momentul transmiterii procesului şi cel al terminării execuției; include timpul necesar execuției, plus timpul petrecut pentru a aştepta la diverse resurse, inclusive procesorul. Este potrivit mai ales pentru sistemele cu prelucrare în loturi.
• Timpul de raspuns - este indicat pentru sistemele interactive şi reprezintăn intervalul de timp dintre momentul lansării procesului şi cel al terminării lui.
• Termenul limită la care un proces trebuie executat. Când este prevăzut, celelalte criterii sunt subordonate**.**

Criteriile orientate către sistem sunt:
• Productivitatea - reprezintă numărul de procese executate pe unitatea de timp.
• Gradul de utilizare al procesorului - procentul de timp în care procesorul este ocupat.

• Evitarea înfometării - timpul de aşteptare al proceselor pentru a fi servite de processor este suficient de mic.
• Utilizarea priorităților - procesele cu prioritate mai înaltă să fie servite preferențial de CPU.
• Încărcarea (Utilizarea) echilibrată a resurselor - resursele să nu fie ocupate pentru un interval de timp mare (sunt implicate şi celelalte strategii de planificare).

Structura planificatorului. În figura 2.2.1 sunt scoase în evidență componentele planificatorului precum şi conexiunile dintre ele.

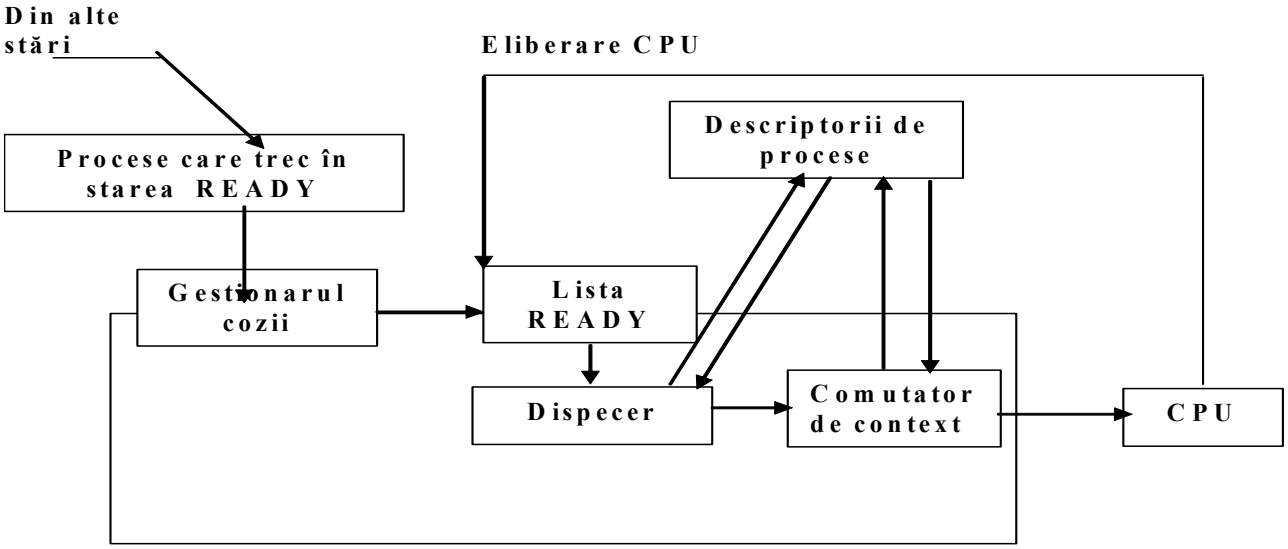


Figura 2.2.1. Structura planificatorului

Gestionarul cozii proceselor care aşteptă să primească serviciile CPU. Când un proces trece în starea READY, descriptorul său este prelucrat şi este introdus într-o coadă înlănțuită numită **lista READY**; această coadă este organizată pe priorități, care sunt calculate de către gestionarul cozii, ori la introducerea procesului în coadă sau în momentul selectării unui proces pentru a fi servit de către CPU.

Comutatorul de context (CS-Context **S**witcher**).** În momentul când planificatorul comută CPU de la execuția unui procesului, la un alt proces, salvează conținuturile tuturor regiştrilor CPU (PC, IR, starea procesului, starea ALU etc.) relative la procesul în execuție. Există două modalități prin care se execută această comutare de context:
- voluntară, care presupune că procesul care deține CPU, cere el însuşi eliberarea acesteia pentru a fi acordată unui alt proces.
- involuntară, care presupune că eliberarea CPU se face prin intermediul unei întreruperi. Această metodă presupune existența unui **dispozitiv de măsurare a intervalelor de timp**, care să genereze o întrerupere oricând expiră intervalul de timp alocat procesului. O astfel de metodă de planificare se mai numeşte şi planificare cu evacuare sau cu forțare.

Salvarea contextului procesului. Când CPU este multiplexată, apar două tipuri de comutare a contextului. În primul caz, se realizează salvarea contextului procesului care este în curs de execuție de către SO şi este încărcat cel al dispecerului. A doua situație constă în cedarea CPU de către dispecer şi încărcarea procesului care urmează să fie servit de către CPU. Salvarea contextului este o operație destul de costisitoare, care poate afecta performanțele sistemului. Dacă notăm cu n, numărul regiştrilor generali şi cu m numărul regiştrilor de stare şi dacă presupunem că pentru a salva un singur registru sunt necesare b operații, fiecare instrucțiune de stocare necesitând K unități de timp, atunci pentru salvarea stării procesului sunt necesare $(n+m)bK$ unități de timp.

Dispecerul este cerut pentru a selecta un nou proces aflat în lista READY, pentru a fi servit de către CPU, indiferent de modalitatea de eliberare a CPU. Acest lucru se poate realiza în două moduri:
• Procesul eliberează în mod voluntar CPU, când face o cere de resursă, trecând astfel controlul administratorului resurselor; CPU fiind eliberată, alte procese o pot utiliza. După terminarea utilizării resursei cerute, procesul revine în starea READY. Dacă resursa cerută nu este disponibilă, procesul este blocat (starea WAIT), până când resursa devine liberă, iar după alocarea şi utilizarea ei trece în starea READY.
• Procesul poate fi evacuat în mod involuntar din utilizarea procesorului.

Performanţe. Planificatorul are o influenţă majoră asupra performanţelor unui calculator, care lucrează în regim de multiprogramare, deoarece el decide când un proces este servit de către CPU. Dacă momentul când un proces este selectat să fie servit, tinde către momentul când este introdus în lista READY, atunci putem spune că el va fi servit aprope când doreşte. În schimb, dacă un proces este neglijat de către dispecer(fenomenul de „înfometare"), atunci va sta destul de mult în starea READY, deci timpul său de execuţie va fi destul de mare. De asemenea, performanţele planificatorului sunt influenţate de timpul necesar pentru comutarea contextului. Toate aceste aspecte sunt influenţate atât de componentele hardware, cât şi de cele software, reprezentate de componenta de planificare a SO.

Notaţii.În cele ce urmează vom nota cu $P = \{p_0,.., p_{n-1}\}$ mulţimea celor *n* procese din sistem, $S(p_i)$ este starea procesului $p_i, i = 0,..,n\text{-}1, S(p_i) \in \{RUN, READY, WAIT\}$. De asemenea $\tau(p_i)$ este timpul efectiv de servire, adică intervalul de timp cât procesul p_i va fi în starea RUN; $W(p_i)$ este intervalul de timp petrecut de proces în starea READY pâna la prima tranziţie în starea RUN. De asemenea, $T(p_i)$ reprezintă timpul total petrecut în sistem de proces, din momentul când procesul intră pentru prima dată în starea READY, până când este executat.

Partiţionarea unui proces în procese mai mici. Presupunem că un proces intercalează efectuarea de calcule, cu cereri de efectuare a unor operaţii de I/O, astfel încât el solicită CPU pentru k periode de timp distincte şi, deci va executa k cereri de efectuare a unor operaţii de I/O. Atunci, timpul total de servire cât îi este alocată CPU va fi $\tau(p_i) = \tau_1 + ... + \tau_k$
De asemenea, dacă $d_1, .., d_k$ sunt timpii necesari efectuării operaţiilor de I/O, atunci timpul cât procesul va sta în sistem va fi $\tau_1 + d_1 + ... + \tau_k + d_k$
Presupunem că procesul p_i este descompus în k procese $p_{i1}, ..., p_{ik}$, în care τ_{ij} este timpul de servire pentru p_{ij}. De asemenea, fiecare proces p_{ij} este executat printr-o prelucrare care nu este întreruptă de alte procese. Reamintesc că orice proces poate fi orientat către efectuarea de calcule sau de operaţii de I/O. O astfel de descompunere urmăreşte ca procesele mai mici să fie orientate spre calcule sau spre operaţii de I/O, în scopul de a se evita cât mai mult operaţiile costisitoare de comutare.

Un model simplificat de planificare a proceselor. Cererile de servicii adresate unităţii centrale se intercalează cu cereri de alocări de resurse. Pe baza consideraţiilor anterioare, modelul de planificare al proceselor poate fi simplificat, neluând în consideraţie efectele datorate competiţiei pentru alocarea resurselor, exceptând CPU(figura 2.2.2). În cadrul acestui model, procesul poate fi numai într-una din stările RUN sau READY.

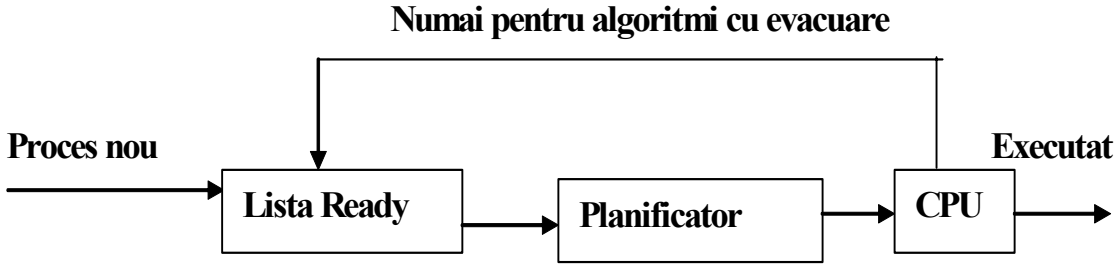


Figura 2.2.2. Structura planificatorului simplificat

Să ne reamintim...
Planificariea execuției proceselor poate fi împărțită în planificare pe termen lung, planificare pe termen mediu şi planificare pe termen scurt.

Planificare pe termen lung se referă la ce programe sunt admise de sistem pentru a fi prelucrate.

Planificarea pe termen mediu se referă la ce procese urmează să concureze pentru a fi servite de CPU.

Planificarea pe termen scurt are ca obiectiv principal alocarea proceselor timpul CPU, astfel încât să se optimize unul sau mai multe aspecte ale funcționării sistemului de calcul.

Planificatorul este format din: gestionarul cozii proceselor care aşteptă să primească serviciile CPU, comutatorul de context şi dispecer.

I. Răspundeți la următoarele întrebări.
1. Care dintre tipurile de planificare controlează gradul de multiprogramare al sistemului.
2. Din ce motive este necesară planificarea pe termen mediu.
3. Care dintre tipurile de planificare urmareşte modalitățille de alocare a timpului CPU proceselor.
4. În cadrul planificării pe termen scurt, care criteriu de eficiență este specific sistemelor interactive.
5. Care este diferența dintre gestionarul cozii proceselor şi dispecer.
6. Ce este comutarea de context? În cazul căror metode de planificare capătă o importanță majoră comutarea de context..
7. În cadrul modelului simplificat de planificare, în ce stări se poate afla un proces.

M2.U2.4. Strategiile fără evacuare

Aceste strategii permit unui proces ca atunci când îi este alocată CPU să fie executat complet, deci nu mai există procesul de comutare şi nici cel de introducere şi extragere din lista READY. Aceste metode se bazează pe modelele de aşteptare.

Algoritmul FCFS(First Come First Served) sau **FIFO (First In First Output)**. Conform acestui algoritm, lucrările sunt servite în ordinea lor cronologică în care cer procesorul. Dispecerul va selecta lucrarea pentru care timpul de aşteptare este cel mai lung. Lista READY este organizată pe principiul FIFO, administratorul cozii adăugând lucrări la sfârşitul cozii, iar dispecerul scoate lucrările pentru a fi servite de CPU din capul cozii. Este un algoritm simplu, dar nu foarte eficient.

Exemplu. Să presupunem că există 5 joburi în sistem, p_0, p_1, p_2, p_3, p_4 cu timpii de execuție 350, 125, 475, 250 respectiv 75. Dacă acestea sosesc în ordinea prezentată, ordinea temporală a execuției lor, este reliefată prin diagrama Gantt din figura 2.2.3. Avem:

$$T(p_0) = \tau(p_0) = 350$$
$$T(p_1) = \tau(p_1) + T(p_0) = 125 + 350 = 475$$
$$T(p_2) = \tau(p_2) + T(p_1) = 475 + 475 = 950$$
$$T(p_3) = \tau(p_3) + T(p_2) = 250 + 950 = 1200$$
$$T(p_4) = \tau(p_4) + T(p_3) = 75 + 1200 = 1275$$

Media timpilor petrecuți în sistem va fi deci

$$T = (350 + 475 + 950 + 1200)/5 = 850$$

De asemenea timpii de așteptare vor fi: $W(p_0) = 0$, $W(p_1) = T(p_0) = 350$, $W(p_2) = T(p_1) = 475$, $W(p_3) = T(p_2) = 950$, $W(p_4) = T(p_3) = 1200$

Deci media timpului de așteptare până la servire, va fi $W = (0 + 350 + 475 + 950 + 1200)/5 = 595$.

Dacă ordinea de sosire este p_4, p_1, p_2, p_3, p_0 vom avea

$$T(p_4) = \tau(p_4) = 75$$
$$T(p_1) = \tau(p_1) + T(p_4) = 125 + 75 = 200$$
$$T(p_2) = \tau(p_2) + T(p_1) = 200 + 475 = 675$$
$$T(p_3) = \tau(p_3) + T(p_2) = 250 + 675 = 925$$
$$T(p_0) = \tau(p_0) + T(p_3) = 75 + 925 = 1000$$

Media timpilor petrecuți în sistem va fi deci

$$T = (75 + 200 + 675 + 925 + 1000)/5 = 2875/5 = 575$$

De asemenea timpii de așteptare vor fi: $W(p_4) = 0$, $W(p_1) = T(p_4) = 75$, $W(p_2) = T(p_1) = 200$, $W(p_3) = T(p_2) = 675$, $W(p_1) = T(p_3) = 925$

Deci media timpului de așteptare până la servire, va fi $W = (0 + 75 + 200 + 675 + 925)/5 = 1875/5 = 355$

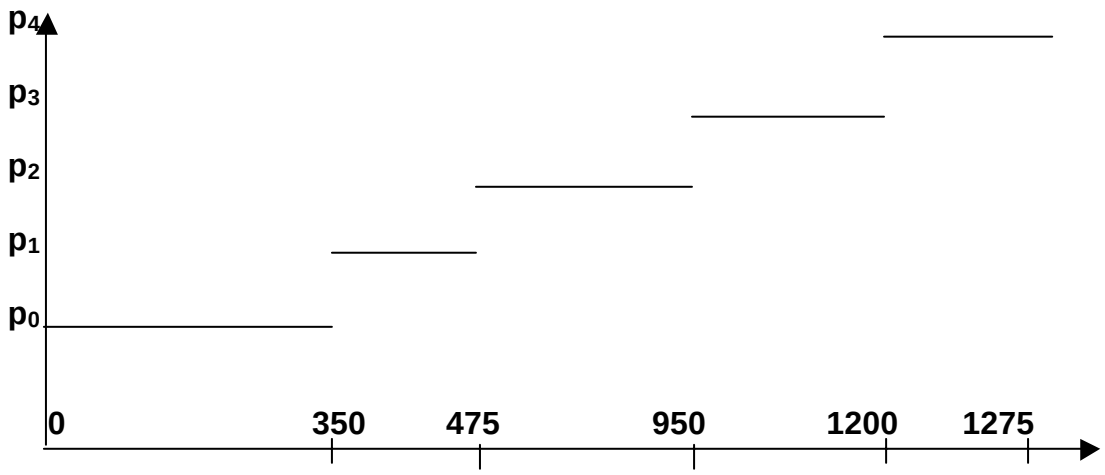


Figura 2.2.3. Ordinea execuției proceselor folosind strategi FCFS

Observație. Mediile timpilor petrecuți în sistem precum şi a timpilor de aşteptare sunt mai mici în cazul al doilea. Mai mult, aceste medii descresc, dacă ordinea de mărime a timpilor de servire respectă ordinea de sosire.

Algoritmul SJN (Shortest Job Next) se bazează pe observația anterioară. De fiecare dată, jobul care se execută primul este cel care consumă cel mai puțin timp CPU. Probabil că acest algoritm este cel mai bun, însă are dezavantajul că ar trebui să se cunoască dinainte timpul CPU necesar al fiecărui proces.

Exemplu. Considerând procesele prezentate în exemplul anterior, conform acestei metode ordinea de execuție este p_4, p_1, p_3, p_0, p_2 , reliefată în diagrama Gantt din figura 2.2.4 şi atunci:

$$T(p_4) = \tau(p_4) = 75$$
$$T(p_1) = \tau(p_1) + T(p_4) = 125 + 75 = 200$$
$$T(p_3) = \tau(p_3) + T(p_1) = 250 + 200 = 450$$
$$T(p_0) = \tau(p_0) + T(p_3) = 350 + 450 = 800$$
$$T(p_2) = \tau(p_2) + T(p_0) = 475 + 800 = 1275$$

Media timpilor petrecuți în sistem va fi deci
$T = (75 + 200 + 450 + 800 + 1275)/5 = 2450/5 = 490$

De asemenea timpii de aşteptare vor fi: $W(p_4) = 0$, $W(p_1) = T(p_4) = 75$,
$W(p_3) = T(p_1) = 200$, $W(p_0) = T(p_3) = 450$, $W(p_2) = T(p_0) = 800$

Deci media timpului de aşteptare până la servire, va fi
$W = (0 + 75 + 200 + 450 + 800)/5 = 1525/5 = 305$.

Se observă că valorile calculate ale factorilor de eficiență sunt mult mai mici dacă se foloseşte această metodă față de cea anterioară.

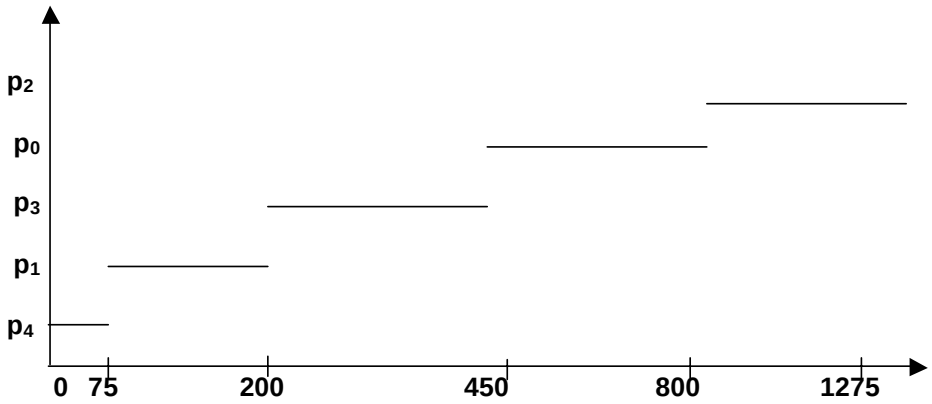


Figura 2.2.4. Ordinea temporală a execuției proceselor folosind strategia SJN

Algoritmul bazat pe priorități. În funcție de importanța lor, job-urile sunt împărțite în clase de priorități. Fiecare job are asociată o prioritate. Aceste priorități sunt numere naturale, un proces având prioritate față de altul dacă numărul alocat are o valoare mai mică. Toate joburile care au asociată o aceeaşi prioritate formează o clasă. Prioritățile job-urilor sunt calculate de administratorul cozii, atunci când intră în lista READY. Joburile se ordonează după aceste priorități, apoi se execută în această ordine. În cadrul aceleiaşi clase se aplică disciplina FIFO.

Se disting două tipuri de priorități: **interne** şi **externe**. Prioritățile interne sunt acordate pe baza unor particularități ale procesului, relative la mediului de calcul respectiv, cum ar fi de

exemplu **timpul de execuție**. Priorităţile externe reflectă **importanţa sarcinii** pe care procesul respectiv o execută, stabilită pe baza anumitor informaţii cum ar fi de, exemplu **numele utilizatorului**(cui aparţine procesul respectiv), **natura sarcinii**(cât de importantă este problema pe care o rezolvă) etc.

Exemplu. Presupunem că avem 5 procese p_0, p_1, p_2, p_3, p_4 ale căror priorităţi sunt 5,2,3,1,4 iar timpii de execuţie sunt cei din exemplul anterior. Diagrama Gantt din figura 2.2.5 descrie ordinea execuţiei proceselor.

Vom avea:

$$T(p_0) = \tau(p_0) + \tau(p_4) + \tau(p_2) + \tau(p_1) + \tau(p_3) = 350 + 75 + 475 + 125 + 250 = 1275$$

$$T(p_1) = \tau(p_1) + \tau(p_3) = 125 + 250 = 375$$

$$T(p_2) = \tau(p_2) + \tau(p_1) + \tau(p_3) = 475 + 125 + 250 = 850$$

$$T(p_3) = \tau(p_3) = 250$$

$$T(p_4) = \tau(p_4) + \tau(p_2) + \tau(p_1) + \tau(p_3) = 75 + 475 + 125 + 250 = 925$$

Deci media timpului de aşteptare va fi $T = (1275 + 375 + 850 + 250 + 925)/5 = 735$

De asemenea, timpii de aşteptare se pot detemina într-un mod asemănător celui din exemplul anterior, aceştia fiind: $W(p_0)=925$, $W(p_1)=250$, $W(p_2)=375$, $W(p_3)=0$, $W(p_4)=850$. Deci media timpului de aşteptare va fi $W(p_0)=(925+250+375+0+850)/5=480$

Alegerea priorităţilor pentru procese este o problemă importantă, pentru ca această metodă de planificare să fie eficientă. Există pericolul apariţiei fenomenului de „înfometare”, adică să existe o mulţime nevidă de procese care au o prioritate slabă pentru un interval de timp suficient de mare şi deci, care să aştepte un timp prea mare până când vor fi executate. Pentru a se evita acest fenomen, se poate considera drept criteriu de acordare a priorităţilor timpul de când jobul respectiv se află în sistem.

Toate celelalte strategii folosite, sunt cazuri particulare ale acestora. În cadrul strategiei FIFO, priorităţile sunt alocate în funcţie de ordinea sosirii în sistem, pe când în cazul SJN priorităţile sunt alocate în funcţie de dimensiunea timpului procesor necesar execuţiei job-ului respectiv.

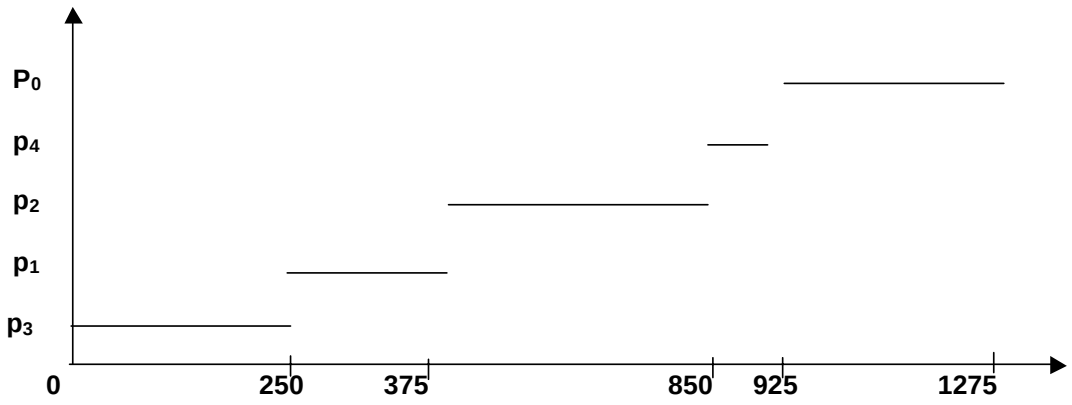


Figura 2.2.5. Ordinea temporală a execuţiei proceselor folosind strategia bazată pe priorităţi

Să ne reamintim...
Strategiile fără evacuare permit unui proces ca atunci când îi este alocată CPU să fie executat complet.
Conform algoritmului FCFS, lucrările sunt servite în ordinea lor cronologică în care cer procesorul.
Algoritmul SJN presupune execuția job-urilor în ordinea crescătoare a timpilor lor de execuție.
Algoritmul bazat pe priorități presupune împărțirea lucrărilor în clase de priorități și execuția lor conform apartenenței la acestă clasă. Celelalte metode sunt cazuri particulare ale acesteia.

I. Răspundeți la următoarele întrebări.
1. Cărui tip de prelucrare sunt specifice strategiile fără evacuare.
2. Care este dezavantajul metodei FCFS.
3. Care sunt avantajele şi dezavantajele metodei SJN.
4. Care sunt avantajele metodei bazată pe împărțirea job-urilor în clase de priorități.
5. Cum se poate evita apariția fenomenului de înfometare în cazul folosirii metodei bazată pe împărțirea job-urilor în clase de priorități.
6. Justificați de ce metoda bazată pe împărțirea job-urilor în clase de priorități este o generalizare a celorlalte metode.
II. 1. Presupunem că într-un sistem cu un singur procesor sosesc 5 job-uri p_0, p_1, p_2, p_3, p_4 şi timpii de servire sunt 80, 20, 10, 20, 50.
i) Dacă strategia de servire a joburilor este FCFS iar ordinea este cea specificată anterior, să se descrie ordinea temporală a execuției cu ajutorul unei diagrame Gantt şi să se determine factorii de eficiență ai sistemului. Discuție.
ii) Aceleași cerințe dacă politica de servire este SJN. Discuție.
2. Dacă sistemul foloseşte planificarea bazată pe priorități, în care prioritatea este implementată prin asocierea unui număr natural fiecărui job, în care un număr mai mic înseamnă o prioritate mai înaltă şi secvența de job-uri şi timpii de execuție sunt cei din problema anterioră, iar prioritățile asociate job-urilor sunt 3, 1, 4, 5, 2 să se descrie ordinea execuției proceselor cu ajutorul unei diagrame Gantt şi să se determine factorii de eficiență ai sistemului.

M2.U2.5. Strategii cu evacuare.

În cazul algoritmilor cu evacuare, CPU este alocată procesului cu prioritatea cea mai înaltă, dintre toate procesele care sunt în starea READY. Toate procesele cu prioritatea mai mică cedează CPU procesului cu prioritatea cea mai înaltă, atunci când acesta o cere. Oricând un proces intră în starea READY, el poate întrerupe imediat procesul servit de către CPU, dacă acesta are o prioritate mai slabă. Aceasta înseamnă că planificatorul este apelat de fiecare dată când un proces intră în starea READY. De asemenea, el declanşează măsurătorul de intervale de timp atunci când cuanta de timp a trecut.

Strategiile cu evacuare sunt adesea folosite pentru a se asigura un răspuns rapid proceselor cu o prioriate înaltă şi pentru a se asigura o bună partajare a CPU între toate procesele. Există versiuni cu evacuare pentru unele dintre strategiile fără evacuare prezentate anterior. În cazul strategiei SJN, procesului cu cererea de timp cea mai mică îi este alocat CPU. Dacă p_i este în curs de execuție şi intră în starea READY un alt proces, p_j, atunci prin metoda SJN este

necesară compararea lui $\tau(p_i)$ cu $\tau(p_j)$. Acest lucru este posibil deoarece se cunoşte că timpul de servire al lui p_i este cel mai mic, dintre timpii tuturor proceselor aflate în starea READY. Dacă p_j cere un timp de execuţie mai mic decât timpul rămas până la terminarea execuţiei lui p_i, atunci lui p_j i se va aloca CPU, iar p_i va fi introdus în lista READY, cu $\tau(p_i)$ fiind timpul rămas până la terminarea execuţiei lui. Într-un mod similar, se procedează şi în cazul strategiei bazată pe împărţire proceseloe gata de execuţie în clase de priorităţi.

În cazul algoritmilor fără evacuare, nu am considerat costul schimbării de context între procese, deoarece atunci am presupus că odată ce unui proces îi este alocată CPU, acesta va fi executat până la capăt. Aceată problemă capătă o importanţă deosebită în cazul strategiilor cu evacuare, datorită faptului că pe durata execuţiei unui proces acesta poate fi evacuat de mai multe ori, iar fiecare evacuare este însoţită de o schimbare de context, al cărei cost nu poate fi neglijat.

Planificarea circulară(RR-**R**ound **R**obin**)** este poate cea mai utilizată dintre toţi algoritmii de planificare, în special de SO care lucrează în time-sharing. Principala ei caracteristică, este servirea echitabilă a tuturor proceselor care cer alocarea CPU. Să presupunem că în sistem avem n procese $p_0, .., p_{n-1}$, ordonate după indicele lor. Se defineşte o cuantă de timp, dependentă de sistem şi pe durata unei cuante se alocă procesorul unui proces. Coada READY a proceselor este tratată circular; mai întâi este servit p_0, apoi p_1, ş.a.m.d p_{n-1}, după care este servit din nou p_0. Dacă procesorul termină de servit un proces înainte de expirarea cuantei de timp alocate, o nouă cuantă de timp este definită şi aceasta este alocată procesului următor. Când se termină execuţia unui proces, el este scos din lista READY. Când un nou proces este introdus în lista READY, este inserat ultimul în această coadă circulară. De fiecare dată când apare un astfel de eveniment, planificatorul este apelat imediat, şi eventual se defineşte o nouă cuantă de timp şi se selectează un nou proces.

În cazul acestui algoritm, trebuie să luăm în consideraţie efectul schimbării de context între procese. Fie C timpul necesr efectuării acestei operaţii. Dacă fiecăruia dintre cele n procese îi sunt alocate q unităţi de timp CPU, atunci timpul total de servire al celor n procese va fi $n(q+C)$.

Exemplu. Să considerăm procesele p_0, p_1, p_2, p_3, p_4 care au timpii de execuţie definiţi în exemplele anterioare, iar cuanta de timp este de 50 de unităţi şi presupunem că se neglijează timpul necesar schimbării de context. În figura 2.2.6 este prezentată ordinea servirii proceselor şi momentele terminării execuţiei lor.
Din diagrama din figura 2.2.6 putem deduce:
-Timpii petrecuţi în sistem de fiecare proces: $T(p_0)=1100$, $T(p_1)=550$, $T(p_2)=1275$, $T(p_3)=950$, $T(p_4)=475$ şi media timpului petrecut în sistem de un proces $T=(1100+550+1275+950+475)/5=870$.
-Timpii de aşteptare până la prima servire: $W(p_0)=0, W(p_1)=50, W(p_2)=100, W(p_3)=150, W(p_4)=200$ şi media acestor timpi *W=(0+50+100+150+200)/5=100.*
-Dacă considerăm şi timpul de comutare între procese, ca fiind de 10 unităţi de timp, obţinem: $T(p_0)=1320, T(p_1)=660, T(p_2)=1535, T(p_3)=1140, T(p_4)=565, T=(1320+660+1535+1140+565)/5=1044, W(p_0)=0$, $W(p_1)=60$, $W(p_2)=120$, $W(p_3)=180$, $W(p_4)=220$ şi $W=(0+60+120+180+240)/5=120$.

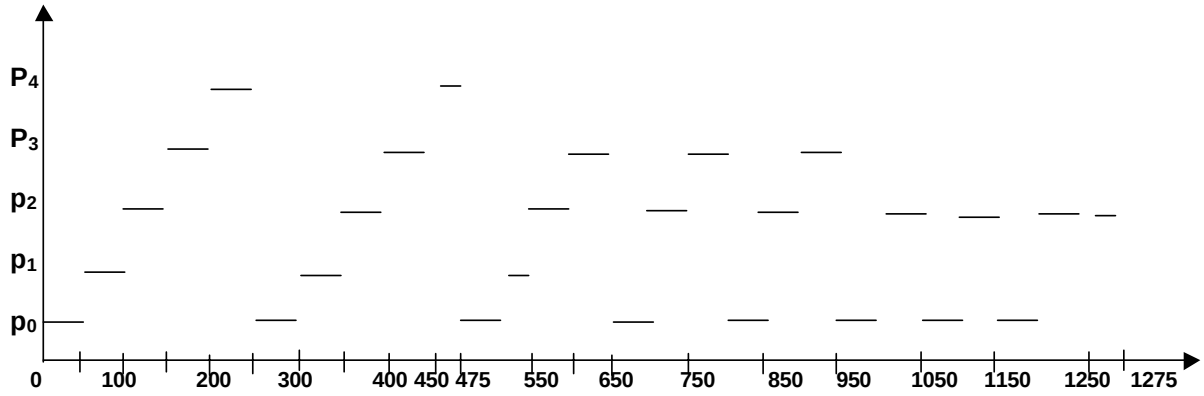


Figura 2.2.6. Planificare RR cu cuanta de timp de 50 de unități

Metoda cozilor pe mai multe niveluri este o extensie a planificării bazate pe priorități, care presupune că toate procesele care au aceeaşi prioritate sunt plasate în aceeaşi coadă. Între cozi, planificatorul alocă CPU folosind o anumită strategie, iar în interiorul cozii o altă strategie. De exemplu, la un SO de tip mixt, interactiv şi serial, pot fi create 5 cozi distincte: **taskuri sistem**, **lucrări interactive**, **lucrări în care se editează texte**, **lucrări seriale obişnuite**, **lucrări seriale ale studenților** etc. Aceste cozi se bazează pe împărțirea proceselor (lucrărilor) în clase de priorități; în cadrul fiecărei clase se poate utiliza o anumită strategie (FIFO, SJN, RR).

Să ne reamintim...

În cazul algoritmilor cu evacuare, CPU este alocată procesului cu prioritatea cea mai înaltă, dintre toate procesele care sunt în starea READY.

Strategiile cu evacuare sunt adesea folosite pentru a se asigura un răspuns rapid proceselor cu o prioriate înaltă sau pentru a se asigura o bună partajare a CPU între toate procesele.

Metoda cozilor pe mai multe niveluri este o extensie a planificării bazate pe priorități, care presupune că toate procesele care au aceeaşi prioritate sunt plasate în aceeaşi coadă

I. Răspundeți la următoarele întrebări.

1. Cum se poate implementa strategia SJN folosind o strategie cu evacuare.

2. Planificarea execuției proceselor bazată politica de servire RR(Round Robin) presupune acordarea unei cuante fixe de timp fiecărui proces. Care sunt avantajele folosirii unei cuante de dimensiune mică, respectiv a unei cuante de dimensiune mare.

3 Metoda RR presupune existența unei liste a proceselor din sistem, în care fiecare proces apare o singură dată în listă. Ce se poate întâmpla dacă un proces apare de două sau mai multe ori? Argumentați de ce ar trebui să se permită acest lucru.

II. i) Dacă în sistem sunt lansate în acelaşi timp 5 procese p_0, p_1, p_2, p_3, p_4 şi timpii de servire sunt 75, 40, 25, 20, 45, iar politica de servire este RR, cu cuanta de timp alocară unui job de 15, fără a se considera timpul necesar comutării de context, să se descrie ordinea temporală a execuției cu ajutorul unei diagrame Gantt şi să se determine factorii de eficiență ai sistemului.

ii) Aceleaşi cerințe, dacă presupunem că se ia în considerație costul comutării de context, acesta fiind de 5 unități.

M2.U2.6. Test de evaluare a cunoştinţelor

I. . Selectaţi varianta corectă.

1. De fiecare dată când se termină execuţia unui proces, dacă procentul de timp de lenevire al CPU depăşeşte un anumit prag, planificatorul poate decide:

a) Să distribuie mai multe resurse proceselor în execuţie ☐

b) Să creeze încă unul sau mai multe procese . ☐

c) Să aloce mai multă memorie proceselor în execuţie ☐

d) Să aloce mai mult spaţiu pe disc proceselor în execuţie ☐

2. Planificare pe termen lung se referă la:

a) Ce programe sunt admise de sistem pentru a fi prelucrate. ☐

b) Ce zone de memorie sunt alocate. ☐

c) Ce fişiere sunt salvate. ☐

d) Ce periferice vor fi alocate. ☐

Planificarea pe termen mediu se poate realiza din diverse motive:

3. Trecerea din starea **new** în starea **ready** se referă la:

a) Planificarea pe termen lung. ☐

b) Planificarea pe termen mediu ☐

c) Planificarea pe termen scurt. ☐

d) Planificarea pe termen nedefinit ☐

4. Obiectivul principal al planificării pe termen scurt este de:

a) A aloca proceselor timpul CPU astfel încât să se optimize spaţiul de pe discuri. ☐

b) A aloca proceselor timpul CPU astfel încât să se optimize consumul de memorie internă. ☐

c) A aloca proceselor timpul CPU astfel încât să se optimize timpii necesari schimbului de informaţii între procese. ☐

d) A aloca proceselor timpul CPU astfel încât să se optimize unul sau mai multe aspecte ale funcţionării sistemului de calcul. ☐

5. Care dintre componentele planificatorului prelucrează descriptorul unui proces trecut în starea **ready** şi îl introduce în **lista READY**:

a) Dispecerul ☐

b) Gestionarul cozii proceselor ☐

c) Comutatorul de context ☐

d) Încărcătorul ☐

6**.** Care dintre componentele planificatorului este cerut pentru a selecta un nou proces aflat în lista READY

a) Dispecerul ☐

b) Gestionarul cozii proceselor ☐

c) Comutatorul de context ☐

d) Încărcătorul ☐

7. Care dintre componentele planificatorului este necesară reluării corecte a execuţiei unui proces.

a) Dispecerul ☐

b) Gestionarul cozii proceselor ☐

c) Comutatorul de context ☐

d) Încărcătorul ☐

8. Principala ei caracteristică a planificării circulare este:

a) Servirea echitabilă a tuturor proceselor care cer alocarea CPU. ☐

b) Servirea echitabilă a tuturor proceselor nucleului sistemului de operare. ☐

c) Servirea pe bază de priorități a tuturor proceselor care cer alocarea CPU. ☐

d) Servirea pe bază de priorități a tuturor proceselor nucleului sistemului de operare. ☐

9. Strategiile cu evacuare sunt adesea folosite pentru:

a) A se asigura un răspuns rapid proceselor cu o prioriate înaltă şi pentru a se asigura o bună partajare a CPU între toate procesele ☐

b) A se asigura un răspuns rapid proceselor cu o prioriate slabă şi pentru a se asigura o bună partajare a CPU între toate procesele ☐

c) A se asigura un răspuns rapid proceselor cu o prioriate înaltă şi pentru a se asigura o bună partajare a CPU între procesele nucleului sistemului de operare ☐

d) A se asigura un răspuns rapid proceselor cu o prioriate slabă şi pentru a se asigura o bună partajare a CPU între procesele nucleului sistemului de operare ☐

M2.U2.7. Rezumat

Planificatorul este responsabil de multiplexarea CPU de către o mulțime de procese care sunt gata de execuție. Oricând un proces aflat în execuție, doreşte să cedeze CPU sau când este selectat un proces gata de execuție, pentru a primi serviciile CPU, intervine planificatorul pentru a lua decizia potrivită.

Planificarea execuției proceselor este de mai multe tipuri. Planificarea pe termen lung urmăreşte să crească gradul de multiprogramare a sistemului de calcul, prin crearea de noi procese atunci când sistemul dispune de resursele necesare. Planificarea pe termen mediu decide când un proces intră în competiție pentru a primi serviciile CPU. Planificarea pe termen scurt decid alocarea timpului CPU către procese, astfel încât să se optimizeze anumite obiective de eficiență ale sistemului de calcul şi utilizatorului.

Strategiile de planificare pot fi împărțite în strategii fără forțarea, respectiv cu forțarea procesului de a ceda CPU; în acest sens, FCFS, SJN, împărțirea proceselor în clase de priorități, sunt cele mai importante metode utilizate de strategiile fără forțare, pe când RR şi cozile multiple sunt procedeele cele mai utilizate de strategiile cu forțarea procesului de cedare a CPU.

Unitatea de învățare M2.U3. Starea de interblocare

Cuprins

M2.U3.1. Introducere

În cursul execuției lor, procesele solicită diverse resurse ale sistemului de calcul. Alocarea acestora trebuie să se facă după anumite reguli, astfel încât sistemul să nu intre în starea de interblocare. Această situație apare atunci când există o mulțime de procese, astfel încât fiecare proces din această mulțime, pentru a-şi continua execuția, aşteaptă eliberarea unei resurse de către un alt proces care aparține acestei mulțimi.

Există mai multe metode de rezolvare a acestei probleme. Evitarea interblocării presupune că de fiecare dată când se cere de către un proces alocarea de resurse se verifică dacă sistemul intră în starea de interblocare după ce se alocă resursele respective. Detectarea interblocării presupune că pe baza comportamentului sistemului, se deduce că acesta se află în această stare şi apoi se aplică algoritmii necesari determinării proceselor care provoacă această stare a sistemului.

M2.U3.2. Obiectivele unității de învățare

Această unitate de învățare îşi propune ca obiectiv principal o inițiere a studenților în rezolvarea problemei interblocării proceselor. La sfârşitul acestei unități de învățare studenții vor fi capabili să:

- înțeleagă şi să explice problema interblocării proceselor;
- identifice condițiile în care poate apare interblocarea proceselor;
- înțeleagă şi să explice metodele evitării interblocării;
- înțeleagă şi să explice metodele detectării interblocării;
- înțeleagă şi să aplice algoritmii prezentați.

Durata medie de parcurgere a unității de învățare este de 3 ore.

M2.U3.3. Concepte introductive

Resursele logice (fişiere, baze de date, semafoare etc.) sau fizice (imprimante, spaţiul de memorie internă, memorii externe, cicluri UC etc.) pot fi formate din unul sau mai multe elemente. Etapele parcurse de un proces în cursul utilizării unei resurse sunt:
• **cerere de acces**: dacă cererea nu poate fi satisfăcută imediat, procesul care a formulat-o va fi nevoit să aştepte până când poate dobândi resursa;
• **utilizare**: procesul poate folosi resursa;
• **eliberare**: procesul eliberează resursa.
Cererea şi eliberarea de resurse se face prin **apeluri de sistem**. Evidenţa alocării resurselor se realizează prin intermediul unei **tabele de sistem.** Dacă un proces cere o resursă ale cărei elemente sunt alocate altor procese din sistem, atunci procesul care a efectuat cererea va fi pus într-o coadă de aşteptare, asociată resursei respective.

Se spune că un set de procese se află în stare de **interblocare** atunci când orice proces din setul respectiv se află în aşteptarea unui eveniment de eliberare a unei resurse cerute, ce poate fi produs numai de către un proces aflat în mulţimea respectivă. Pentru rezolvarea problemei interblocării se folosesc, în principiu două metode. Prima constă în utilizarea unui protocol care să nu permită niciodată sistemului să intre în starea de interblocare. Acest lucru se realizează fie prin **prevenirea**, fie prin **evitarea** interblocării. Cea de a doua metodă permite sistemului intrarea în starea de interblocare şi apoi rezolvă această problemă.

Condiţii necesare pentru apariţia interblocării.
• **excludere mutuală**: există cel puţin o resursă ocupată în mod exclusiv, adică fără a putea fi folosită în comun de către un singur proces; dacă un alt proces formulează o cerere pentru aceeaşi resursă, va fi nevoit să aştepte până în momentul eliberării ei;
• **ocupare şi aşteptare**: există cel puţin un proces care ţine ocupată cel puţin o resursă şi aşteaptă să obţină resurse suplimentare ocupate în acel moment de către alte procese;
• **imposibilitatea achiziţionării forţate**: resursele nu pot fi achiziţionate forţat de către un proces de la un alt proces care le ocupă în acel moment; resursele pot fi eliberate numai de către procesele care le ocupă, decât după ce acestea şi-au îndeplinit sarcinile;
• **aşteptare circulară**: în sistem există un set de procese aflate în starea de aşteptare, $(p_1, p_2, .., p_n)$, astfel încât p_1 aşteaptă eliberarea unei resurse ocupate de către p_2, p_2 aşteaptă eliberarea unei resurse ocupate de către p_3, ş.a.m.d. p_n aşteaptă eliberarea unei resurse ocupate de către p_1.

Observaţie. Ultima condiţie implică şi cerinţa de ocupare şi aşteptare, astfel încât cele patru condiţii nu sunt complet independente; cu toate acestea este util ca fiecare condiţie să fie discutată şi tratată separat.

Evitarea interblocării. Starea alocării resurselor este definită de numărul de resurse disponibile şi alocate şi de numărul maxim de cereri de resurse formulate de către procese. Se spune că o stare este **sigură** dacă sistemul poate aloca fiecărui proces resursele cerute (până la numărul maxim), într-o anumită ordine şi evitând apariţia interblocării.

Mai exact, sistemul se află într-o stare sigură numai dacă există o **secvenţă sigură**. Se spune că o secvenţă de procese $(p_1, p_2, .., p_n)$ este o secvenţă sigură pentru starea de alocare curentă dacă, pentru fiecare proces p_i, resursele pe care acesta le-ar mai putea cere pot fi alocate dintre resursele disponibile, la care se mai adaugă resursele deţinute de către toate celelalte procese p_j, cu $j<i$. În acest caz, dacă resursele cerute de către procesul p_i nu sunt disponibile imediat, acesta va trebui să aştepte până când toate procesele p_j, cu $j<i$ îşi încheie execuţia. În acest moment p_i poate obţine toate resursele de care are nevoie, îşi

termină sarcina, eliberează resuirsele şi se încheie, după care procesul p_{i+1} poate obține resursele pe care le dorește ș.a.m.d. În cazul în care nu există o astfel de secvență, se spune că starea sistemului este nesigură. O stare de interblocare este o stare nesigură; nu toate stările nesigure sunt interblocări, dar o stare nesigură poate conduce la interblocare. Într-o stare nesigură, sistemul de operare nu poate împiedica procesele să formuleze în așa fel cererile de alocare a resurselor încât acestea să ducă la interblocare.

Exemplu: Să considerăm un sistem în care 4 procese p_1, p_2, p_3, p_4 folosesc în comun o resursă cu 20 de elemente. Pe durata întregii execuții procesele au nevoie de maximum 14, 6, 10 şi respectiv 9 elemente. Inițial, procesele formulează o cerere pentru 7, 3, 4 şi respectiv 2 elemente. Analizând situația sistemului la momentul inițial, se constată că el se află într-o stare sigură, deoarece, de exemplu, secvența p_2, p_1, p_3, p_4 este o secvență sigură.

Justificare. După alocarea inițială, mai rămân 20-16=4 elemente. Pentru a fi executat p_2, există suficiente elemente ale resursei (6<4+3), adică elementele resursei deținute de proces după alocarea inițială, la care se adaugă cele disponibile. După ce procesul p_2 îşi termină execuția, sunt disponibile 7 elemente ale resursei, care adăugate la cele 7 alocate inițial lui p_1 fac posibilă execuția acestuia ș.a.m.d.

Detectarea interblocării. Atunci când sistemul nu utililizează algoritmi de evitare a interblocării, deorece utilizarea lor ar mări costurile, se pot utiliza metodele de detectare şi revenire, care însă, pe lângă cheltuielile datorate utilizării unor structuri de date suplimentare, presupun şi cheltuieli datorate revenirii din starea de interblocare. Problema care se pune, este când să fie apelat algoritmul de detectare a interblocării şi în cazul în care apare această situație, să existe o metodă de ieşire. Dacă acest lucru se face ori de câte ori se formulează o cerere de resurse, efectul este creşterea substanțială a timpului de calcul. O variantă mai puțin costisitoare este apelarea algoritmului la anumite **intervale de timp** sau atunci când **gradul de utilizare a CPU** scade sub un anumit prag (de exemplu 40 la sută), justificarea fiind că o parte dintre procese nu avansează în execuția lor din cauza unei interblocări, deci CPU nu mai este solicitată de acestea şi deci scade gradul de utilizare a CPU.

Să ne reamintim...

Etapele parcurse de un proces în cursul utilizării unei resurse sunt: cerere de acces, utilizare, eliberare.

Se spune că un set de procese se află în stare de interblocare atunci când orice proces din setul respectiv se află în aşteptarea unui eveniment de eliberare a unei resurse cerute, ce poate fi produs numai de către un proces aflat în mulțimea respectivă.

Condiții necesare pentru apariția interblocării: excludere mutuală, ocupare şi aşteptare, imposibilitatea achiziționării forțate: resursele nu pot fi achiziționate forțat de către un proces de la un alt proces care le ocupă în acel moment; resursele pot fi eliberate numai de către procesele care le ocupă, decât după ce acestea şi-au îndeplinit sarcinile,aşteptare circulară.

Prin evitarea interblocării, atunci când un proces cere resurse se verifică dacă după alocare sistemul intră într-o stare nesigură, care poate fi o stare de interblocare.

Detectarea interblocării permite ca sistemul să intre în stare de interblocare şi atunci când se constată existența acesteia se aplică algoritmi de ieşire din această stare.

Legat de exemplul anterior, justificați că dacă procesului p_4 îi sunt alocate inițial mai mult cu 2 elemente, se trece nu numai într-o stare nesigură dar şi într-o stare de interblocare., deoarece numărul de resurse libere, egal cu 2, este insuficient pentru oricare dintre procesele aflate în aşteptare.

M2.U3.4. Evitarea interblocării pentru resurse cu mai multe elemente

Algoritmul de evitare a interblocării, pentru cazul resurselor **cu mai multe elemente** se numeşte **algoritmul bancherului**. Pentru implementarea algoritmului, sunt necesare câteva structuri de date care să codifice starea de alocare a resurselor sistemului. Dacă n este numărul de procese din sistem şi m este numărul de resurse, se definesc:

• D(„Disponibil"): vector de dimensiune m care indică numărul de elemente disponibile ale resurselor: $D(j)$ conţine numărul elementelor disponibile ale resursei r_j, $j=1,..,m$.

• M(„Maxim"): matrice de dimensiune nxm care indică numărul maxim de cereri care pot fi formulate de către fiecare proces; $M(i,j)$ conţine numărul maxim de elemente ale resursei $r_j(j=1,..,m)$ cerute de procesul $p_i(i=1,..,n)$.

• A(„Alocare"): matrice de dimensiune nxm care indică numărul de elemente din fiecare resursă care sunt alocate în mod curent fiecărui proces; $A(i,j)$ conţine numărul de elemente ale resursei $r_j(j=1,..,m)$ alocate procesului $p_i(i=1,..,n)$.

• N(„Necesar"): matrice de dimensiune nxm care indică numărul de elemente ce ar mai putea fi necesare fiecărui proces; $N(i,j)$ conţine numărul de elemente ale resursei $r_j(j=1,..,m)$ de care ar mai avea nevoie procesul $p_i(i=1,..,n)$ pentru a-şi realiza sarcina; evident că

$$N(i,j)= M(i,j)- A(i,j)$$

Fie C_i vectorul cererilor formulate de către procesul $p_i(i=1,..,n)$; $C_i(j)$ este numărul cererilor formulate de procesul p_i din resursa $r_j(j=1,..,m)$; în momentul în care procesul p_i formulează o cerere de resurse, vor fi parcurse următoarele etape:

Pas 1. Dacă $C_i \le N_i$, se execută **pasul 2**, altfel se consideră că a apărut o eroare, deoarece procesul a depăşit cererea maxim admisibilă.

Pas 2. Dacă $C_i \le D$, se execută **pasul 3**, altfel p_i este nevoit să aştepte (resursele nu sunt disponibile).

Pas 3. Se simulează alocarea resurselor cerute de procesul p_i şi starea se modifică astfel:
$D:=D-C_i$; $A_i:=A_i+C_i$; $N_i:=N_i-C_i$.

Pentru verificarea **stării de siguranţă** a sistemului se foloseşte algoritmul următor. Acesta constă în parcurgerea secvenţei de procese şi simularea terminării execuţiei unui proces, adică adăugarea la disponibilul de resurse existent înaintea terminării procesului de la pasul respectiv a resurselor procesului care tocmai şi-a terminat execuţia. Algoritmul foloseşte:

- vectorul de lucru L, de dimensiune m ale cărui componente vor corespunde resurselor sistemului, componenta $L(i)$ va conţine numărul resurselor r_i disponibile la un moment dat.
- un vector T de dimensiune n, care corespunde proceselor şi care va marca procesele parcurse.
- un vector s de dimensiune n; componenta $s(i)$ va conţine indicele (poziţia) din secvenţă a procesului.

```
Pas 1 (Iniţializări).  L=D; citeşte s.
     pentru i=1,..,n; T(i):=0;
Pas 2. (Parcurgerea secvenţei de procese)
     k=0;
     do
          {k=k+1;
           if (N s(k) <=L) then { L:=L+A s(k);   T(s(k)):=1 }
          }
     until k>n or not T(s(k))
```

Pas 3. (Verificare dacă toate procesele din secvență au fost marcate cu `true`). Dacă `T(i)=1` pentru `i=1,..,n`, atunci sistemul se află într-o stare sigură.

Dacă starea de alocare a resurselor rezultată este sigură, se alocă procesului p_i resursele cerute. În caz contrar, procesul p_i este nevoit să aştepte, iar sistemul reface starea de alocare a resurselor existentă înainte de execuția pasului 3.

Exemplu: Să presupunem că în sistem avem 5 procese p_1, p_2, p_3, p_4, p_5 şi 4 resurse r_1, r_2, r_3, r_4, care au 3, 14, 12 respectiv 13 elemente. Se consideră că starea inițială este definită de:

$$A=\begin{pmatrix}0&0&1&2\\1&0&0&0\\1&3&5&4\\0&6&3&2\\0&0&1&4\end{pmatrix},\ M=\begin{pmatrix}0&0&1&2\\1&7&5&0\\2&3&5&6\\0&6&5&2\\0&6&5&6\end{pmatrix},\ N=\begin{pmatrix}0&0&0&0\\0&7&5&0\\1&0&0&2\\0&0&2&0\\0&6&4&2\end{pmatrix},\ D=\begin{pmatrix}1&5&2&0\end{pmatrix}.$$

Să verificăm că sistemul se află într-o stare sigură. Să considerăm secvența $(p_1, p_3, p_4, p_5, p_2)$ şi să arătăm că este sigură. După pasul 1 al algoritmului de verificare a stării de siguranță, vom avea $L=(1\ \ 5\ \ 2\ \ 0)$, $T=(0\ \ 0\ \ 0\ \ 0\ \ 0)$, $s=(0\ \ 2\ \ 3\ \ 4\ \ 1)$. Aplicarea pasului 2, conduce la:

Iterația 1(procesul p_1): $(0\ \ 0\ \ 0\ \ 0)\le(1\ \ 5\ \ 2\ \ 0)$, $T=(1\ \ 0\ \ 0\ \ 0\ \ 0)$, $L=(0\ \ 0\ \ 1\ \ 2)+(1\ \ 5\ \ 2\ \ 0)=(1\ \ 5\ \ 3\ \ 2)$.

Iterația 2(procesul p_3): $(1\ \ 0\ \ 0\ \ 2)\le(1\ \ 5\ \ 3\ \ 2)$, $T=(1\ \ 0\ \ 1\ \ 0\ \ 0)$, $L=(1\ \ 5\ \ 3\ \ 4)+(1\ \ 5\ \ 3\ \ 2)=(2\ \ 8\ \ 8\ \ 6)$.

Iterația 3(procesul p_4): $(0\ \ 0\ \ 0\ \ 2)\le(2\ \ 8\ \ 8\ \ 6)$, $T=(1\ \ 0\ \ 1\ \ 1\ \ 0)$, $L=(0\ \ 6\ \ 3\ \ 2)+(2\ \ 8\ \ 8\ \ 6)=(2\ \ 14\ \ 11\ \ 8)$.

Iterația 4(procesul p_5): $(0\ \ 6\ \ 4\ \ 2)\le(2\ \ 14\ \ 11\ \ 8)$, $T=(1\ \ 0\ \ 1\ \ 1\ \ 1)$, $L=(0\ \ 0\ \ 1\ \ 4)+(2\ \ 14\ \ 11\ \ 8)=(2\ \ 14\ \ 12\ \ 12)$.

Iterația 5(procesul p_2): $(0\ \ 7\ \ 5\ \ 0)\le(2\ \ 14\ \ 12\ \ 12)$, $T=(1\ \ 1\ \ 1\ \ 1\ \ 1)$, $L=(1\ \ 0\ \ 0\ \ 0)+(2\ \ 14\ \ 12\ \ 12)=(3\ \ 14\ \ 12\ \ 12)$.

Pasul 3 $T=(1\ \ 1\ \ 1\ \ 1\ \ 1)$, deci sistemul se află întro stare sigură

În cazul în care procesul p_2 formulează o cerere suplimentară pentru 4, repectiv 2 elemente din r_2 , respectiv r_3, trebuie să se verifice dacă această cerere poate fi satisfăcută. Conform algoritmului bancherului, se verifică îndeplinirea relațiilor:

$C_2 \le N_2$, adică (0,4,2,0) ≤ (0,7,5,0)

$C_2 \le D$, adică (0,4,2,0) ≤ (1,5,2,0)

Cum inegalitățile sunt îndeplinite, se simulează alocarea. Starea sistemului devine:

$$A=\begin{pmatrix}0&0&1&2\\1&4&2&0\\1&3&5&4\\0&6&3&2\\0&0&1&4\end{pmatrix},\ N=\begin{pmatrix}0&0&0&0\\0&3&3&0\\1&0&0&2\\0&0&2&0\\0&6&4&2\end{pmatrix},\ D=\begin{pmatrix}1&1&0&0\end{pmatrix}$$

Pentru a arăta că starea sistemului este sigură, se aplică algoritmul de verificare a siguranței stării pe secvența $(p_1, p_3, p_4, p_2, p_5)$.

Algoritmul bancherului are o complexitate de ordinul $m \times n^2$; pentru evitarea interblocării, în cazul resurselor cu un singur element se utilizează un algoritm de complexitate mult mai mică, care va fi descris în continuare.

Relativ la exemplul anterior:

i) În starea de alocare inițială a resurselor, verificați că secvențele $(p_1, p_4, p_3, p_5, p_2)$ și $(p_1, p_3, p_4, p_2, p_5)$ sunt sigure.

ii) După cererea suplimentară de resurse, verificați siguranța secvențelor: $(p_1, p_3, p_4, p_5, p_2)$ și $(p_1, p_3, p_2, p_4, p_5)$.

iii) Găsiți și alte secvențe sigure, relativ la cele două stări.

Să ne reamintim...

Algoritmul de evitare a interblocării, pentru cazul resurselor cu mai multe elemente se numește algoritmul bancherului. Algoritmul presupune că există o ordine a execuției proceselor. Când un proces cere un număr de elemente dintr-o resursă, sistemul verifică dacă resursele disponibile, la care se adaugă cele eliberate de procesele care au fost executate, la momentul respectiv, sunt suficiente; în caz afirmativ se face alocarea resurselor.

M2.U3.5. Evitarea interblocării pentru resurse cu un singur element

Graful de alocare a resurselor este de forma $G=(N,A)$, în care $N=P \cup R$, $P=\{p_1,..,p_n\}$ fiind mulțimea proceselor iar $R=\{r_1,..,r_m\}$ mulțimea resurselor. Un arc poate fi de forma (p_i, r_j) sau (r_j, p_i); arcul (p_i, r_j), numit **arc cerere,** are semnificația că procesul p_i a cerut un element al resursei r_j, iar arcul (r_j, p_i), numit **arc alocare,** înseamnă că un element al resursei r_j a fost alocat procesului p_i. În cadrul grafului, procesele sunt reprezentate grafic prin cercuri, iar resursele prin pătrate sau dreptunghiuri. Deoarece resursele pot fi formate din mai multe elemente, fiecare element se reprezintă cu ajutorul unui punct plasat în interiorul pătratului sau dreptunghiului respectiv. Atunci când procesul p_i formulează o cerere pentru un element al resursei r_j, se inserează în graful alocare a resurselor un arc cerere. În momentul în care cererea este satisfăcută, arcul cerere este transformat în arc alocare. Atunci când procesul eliberează resursa, arcul alocare este șters din graf.

Exemplu: Considerăm un sistem cu două resurse r_1 și r_2 care au câte un element, și o resursă r_3 cu două elemente în care există două procese, p_1 și p_2. Avem deci: $P=\{p_1,p_2\}, R=\{r_1,r_2,r_3\}$. Dacă $A=\{(p_1,r_1),(r_1,p_2),(r_2,p_1),(r_3,p_1),(r_3,p_2)\}$, reprezentarea grafică a grafului este cea din figura 2.3.1. Mulțimea de arce ilustrează starea proceselor; în acest caz:

- procesul p_1 are alocată câte un element al resurselor r_2 și r_3 și așteaptă obținerea unui resursei r_1;
- procesul p_2 are alocată resursa r_1 și un element al resursei r_3.

Cunoscând definiția grafului de alocare a resurselor, se poate arăta că dacă graful nu conține circuite, în sistem nu există interblocare. Dacă există un circuit, atunci poate să apară interblocarea. Dacă există resurse cu mai multe elemente, atunci existența unui circuit nu implică în mod necesar apariția interblocării, aceasta fiind doar o condiție necesară dar nu și suficientă. Dacă resursele au numai câte un element, atunci existența unui circuit arată că în sistem există interblocare și fiecare dintre procesele care fac parte din circuit se află în această stare.

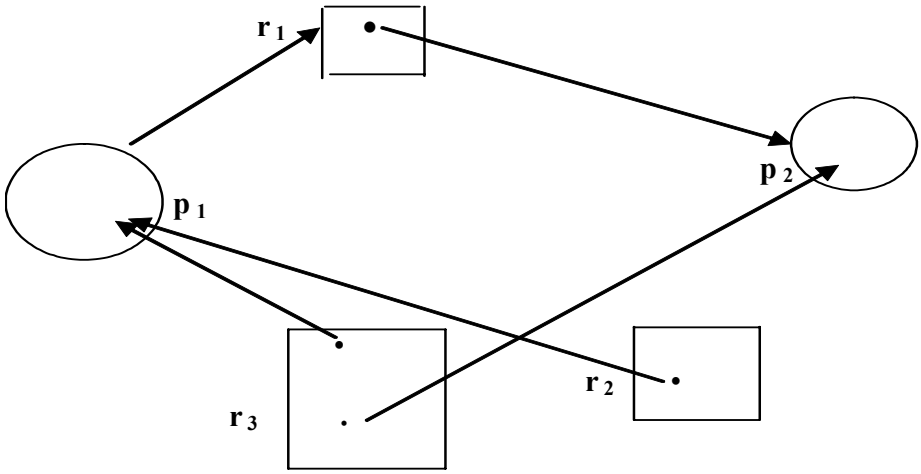


Figura 2.3.1. Graf de alocare a resurselor

Exemplu. În plus față de exemplul anterior, presupunem în plus procesul p_2 formulează o cerere suplimentară, aşa cum este redat în figura 2.3.2. Deoarece resursa de tip r_2 cerută de procesul p_2 nu este disponibilă, fiind deja alocată procesului p_1, în graf se inserează arcul cerere (p_2,r_2), formându-se circuitul $(p_1, r_1, p_2, r_2, p_1)$. În acest moment, cele două procese intră în starea de interblocare, aşteptându-se reciproc pentru eliberarea resurselor de care au nevoie în vederea continuării propriei execuții.

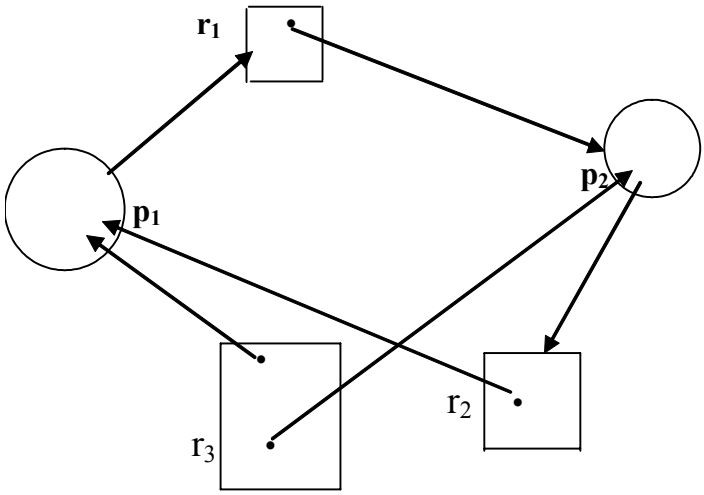


Figura 2.3.2. Graf de alocare a resurselor cu interblocare

Exemplu: Existența unui circuit în graful alocării resurselor nu implică apariția interblocării. În figura 2.3.3 procesele p_1 şi p_2 nu sunt interblocate, deşi fac parte din circuitul (p_1,r_1,p_2,r_2,p_1). În momentul în care p_3 va elibera un element al resursei r_2, acesta va putea fi alocat procesului p_2. Cu alte cuvinte nu există aşteptare reciprocă între procese.

Existența unui circuit în graful alocării resurselor nu implică apariția interblocării. În figura 2.3.3 procesele p_1 şi p_2 nu sunt interblocate, deşi fac parte din circuitul (p_1,r_1,p_2,r_2,p_1). În momentul în care p_3 va elibera un element al resursei r_2, acesta va putea fi alocat procesului p_2. Cu alte cuvinte nu există aşteptare reciprocă între procese.

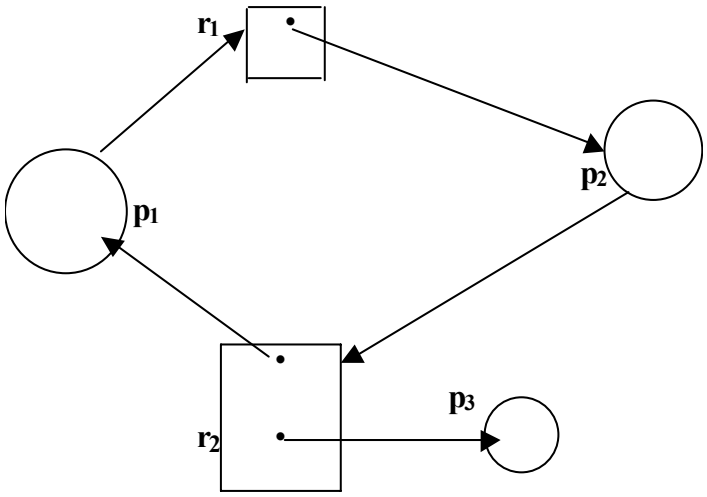


Figura 2.3.3. Graf de alocare a resurselor cu un circuit, fără interblocare

Să ne reamintim...
Evitarea interblocării pentru resurse cu un singur element se bazează pe graful de alocare a resurselor. Dacă există un circuit în acest graf care conține numai resurse cu un singur element, atunci procesele care fac parte din acest circuit sunt în stare de interblocare.

Dați exemple de situații similare celor prezentate, relativ la un sistem cu 4 procese şi 3 resurse.

M2.U3.6. Detectarea interblocării pentru resurse cu mai multe elemente

Algoritmul de detectare pentru resurse cu mai multe elemente foloseşte structurile de date `D`, `A` şi `C` care au aceeşi semnificație ca şi în cazul algoritmului de evitare a interblocării. Algoritmul prezentat în continuare, ia în considerațietoate procesele a căror execuție nu s-a încheiat încă şi analizează toate secvențele posibile de alocare a resurselor pentru aceste procese.

Pas 1. Fie `L` şi `T` doi vectori de dimensioune `m` şi respectiv `n`. Pentru `i=1,..,n` se inițializează `L(i):=D(i)`. Pentru `i=1,..,n`, dacă $A_i \neq 0$, atunci `T(i):=0`, altfel, `T(i):=1`.

Pas 2. Se caută o valoare `i` astfel încât: `T(i)=0` şi $C_i \leq L$. Dacă o astfel de valoare nu există, se execută pasul 4.

Pas 3. Se execută secvența: `L:=L+`A_i; `T(i):=1`; Se execută pasul 2.

Pas 4. Dacă există `i(i=1,..,n)` astfel încât `T(i)=0`, sistemul se află în starea de interblocare. În plus procesul p_i pentru care `T(i)=0`, provoacă interblocarea.

Observație. Motivul pentru care la pasul 3 sunt preluate resursele deținute de procesul p_i, în cazul în care la pasul 2 se îndeplineşte condiția $C_i \leq L$, este că p_i nu este implicat în mod curent într-o interblocare; deci, se poate face presupunerea că p_i nu va mai cere alte resurse pentru a-şi termina execuția şi va elibera toate resursele care i-au fost alocate. În caz contrar, se poate ca mai târziu să apară o interblocare care va fi depistată la următoarea utilizare a algoritmului.

Exemplu: Presupunem că avem trei procese p_1, p_2, p_3, care folosesc în comun trei resurse r_1, r_2, r_3, având câte 4, 2 şi respectiv 2 elemente.
De asemenea, valorile structurilor de date, care dau starea inițială sunt:

$$A=\begin{pmatrix}2 & 2 & 1\\ 1 & 0 & 1\\ 1 & 0 & 0\end{pmatrix},\ C=\begin{pmatrix}0 & 0 & 1\\ 0 & 0 & 0\\ 2 & 1 & 0\end{pmatrix},\ D=\begin{pmatrix}0 & 0 & 0\end{pmatrix}.$$

Să verificăm că nu există interblocare; vom aplica algoritmul prezentat pe secvența (p_2, p_1, p_3).
Pas 1. $T=(0,0,0)$; $L=(0,0,0)$.
Paşii 2 şi 3.
Iterația 1. Pentru procesul p_2 avem $(0\ \ 0\ \ 0)\le(0\ \ 0\ \ 0)$, deci $L:=(1\ \ 0\ \ 1)$, $T:=(0\ \ 1\ \ 0)$
Iterația 2. Pentru procesul p_1 avem $(0\ \ 0\ \ 1)\le(1\ \ 0\ \ 1)$, deci $L:=(1\ \ 0\ \ 1)+(2\ \ 2\ \ 1)=(3\ \ 2\ \ 3)$, $T:=(1\ \ 1\ \ 0)$
Iterația 3. Pentru procesul p_3 avem $(1\ \ 0\ \ 0)\le(3\ \ 2\ \ 3)$, deci $L:=(1\ \ 0\ \ 0)+(3\ \ 2\ \ 3)=(4\ \ 2\ \ 3)$, $T:=(1\ \ 1\ \ 1)$
Pas 4. $T:=(1\ \ 1\ \ 1)$, deci nu există interblocare în sistem.
Dacă procesul p_1 formulează o cerere suplimentară pentru 2 elemente ale lui r_1, modificând prima linie a matricii C, care va conține acum valorile 2,0,1, sistemul intră în starea de interblocare. Chiar dacă procesul p_2 îşi încheie execuția şi eliberează resursele care i-au fost alocate $(1,0,1)$, numărul total de resurse disponibile nu este suficient pentru a putea acoperi necesarul formulat de oricare dintre celelalte două procese. Deci procesele p_1 şi p_3 sunt interblocate.

Algoritmul de detectare a interblocării pentru resurse cu mai multe elemente necesită un număr de operații de ordinul mxn^2. În cele ce urmează, va fi prezentat un algoritm de complexitate mai mică, ce se sprijină pe graful de alocare a resurselor.

Ce se întâmplă dacă procesul p_1 formulează o cerere suplimentară pentru 2 elemente ale lui r_1?
Indicație: prima linie a matricii C va conține acum valorile 2,0,1.

Să ne reamintim...
Algoritmul de detectare a interblocării pentru resurse cu mai multe elemente foloseşte structurile care au aceeşi semnificație ca şi în cazul algoritmului de evitare a interblocării. Algoritmul prezentat ia în considerațietoate procesele a căror execuție nu s-a încheiat încă şi analizează toate secvențele posibile de alocare a resurselor pentru aceste procese.

M2.U3.7. Detectarea interblocării pentru resurse cu un singur element

Dacă toate resursele au câte un singur element, se poate defini un algoritm care utilizează o variantă a grafului de alocare a resurselor, numit graful ”aşteaptă-pentru”. Acesta se obține din graful de alocare a resurselor prin eliminarea nodurilor resursă şi transformarea arcelor corespunzătoare: un arc (p_i, p_j) indică faptul că p_i aşteaptă ca p_j să elibereze resursa care îi este necesară. Într-un graf ”aşteaptă-pentru”, se obține un arc (p_i, p_j) există dacă şi

numai dacă, există o resursă oarecare r_k, astfel încât să existe două arce (p_i, r_k) şi (r_k, p_j) în graful alocării resurselor asociat.

Folosind această metodă, se poate afirma că în sistem există interblocare dacă şi numai dacă graful „aşteaptă-pentru" conţine un circuit. Pentru a detecta interblocările, sistemul trebuie să actualizeze graful „aşteaptă-pentru" şi să apeleze periodic un algoritm care să testeze dacă există circuite în acest graf. Acest algoritm necesită un număr de n^2 operaţii, deci complexitatea este mai mică decât a algoritmului general. Dezavantajul metodei îl reprezintă operaţiile suplimentare necesare reprezentării şi actualizării grafului „aşteaptă-pentru". Pentru a se ieşi din starea de interblocare, se elimină noduri din graf împreună cu arcele adiacente, până când se elimină toate ciclurile. Acest lucru corespunde suspendării proceselor corespunzătoare.

Exemplu: În figura 2.3.4 este prezentat un graf de alocare a resurselor iar în figura 2.3.5 graful "aşteaptă-pentru" asociat. În acest graf avem circuitele: (p_1, p_2, p_3, p_1), (p_2, p_3, p_2). Dacă se suspendă execuţia procesului p_2, adică se elimină nodul corespunzător şi arcele adiacente lui, în graf nu mai există cicluri şi deci se iese din interblocare.

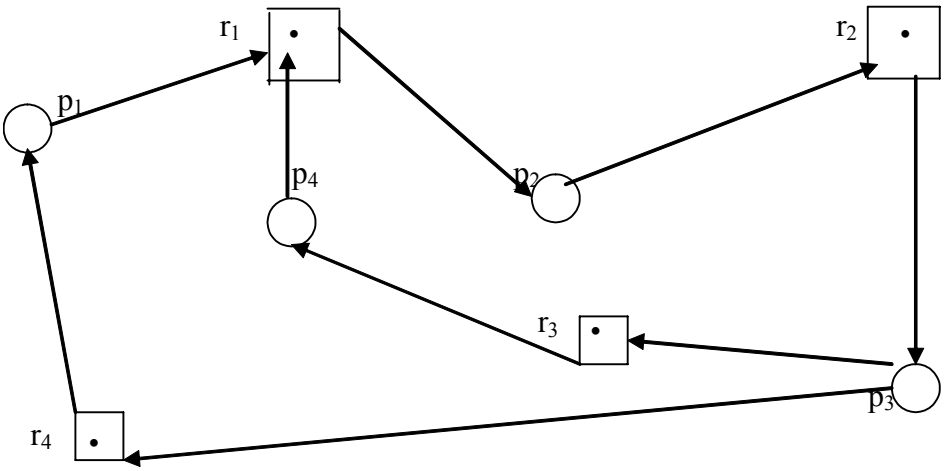


Figura 2.3.4. Graf de alocare a resurselor

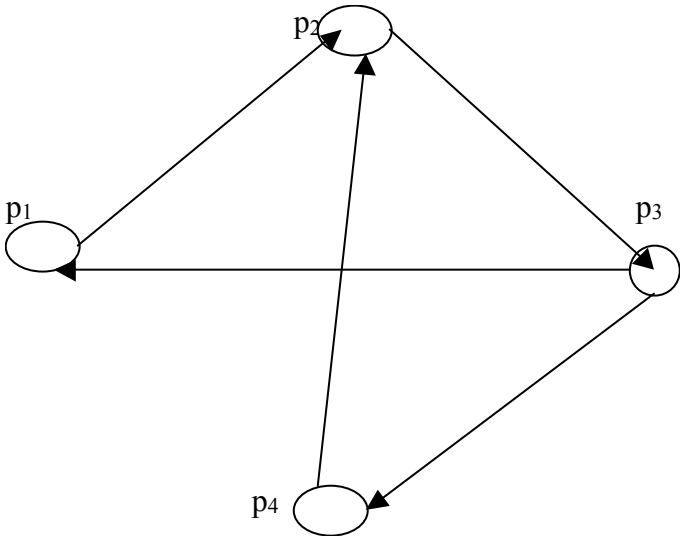


Figura 2.3.5. Graf „aşteaptă-pentru" asociat

Adăugați la graful din figura 2.3.4. o resursă r_5 şi un proces p_5, astfel încât resursa r_5 este alocată procesului p_5 şi cerută şi de p1; p_5 cere şi el r_1. Scrieți graful ”aşteaptă-pentru” corespunzător; specificați dacă există interblocare şi cum se poate ieşi din această stare.

Să ne reamintim...

Pentru detectarea interblocării, în cazul resurselor cu un singur element se utilizează o variantă a grafului de alocare a resurselor, numit graful ”aşteaptă-pentru”. Dacă în acest graf există un circuit, atunci în sistem există interblocare generată de procesele care fac parte din acest circuit.

M2.U3.8. Alte metode de rezolvare a problemei interblocării

Achiziționarea forțată a resurselor. Această metodă asigură eliminarea stării de interblocare prin achiziționarea succesivă a resurselor utilizate de anumite procese şi alocarea lor altor procese, până când se elimină interblocarea. Principalele probleme care trebuie rezolvate în acest caz sunt:
• **alegerea proceselor** cărora li se vor lua resursele, după criterii care să conducă la costuri minime(numărul resurselor ocupate, timpul de execuție consumat);
• **reluarea execuției**: un proces căruia i s-au luat forțat anumite resurse nu-şi mai poate continua normal execuția, ea trebuie reluată dintr-un moment anterior, când a primit prima dintre resursele luate;
• **evitarea „înfometării”** , adică să nu fie selectat pentru achiziționare forțată a resurselor un acelaşi proces.

Metode mixte de tratare a interblcărilor. Practica a dovedit că nici una dintre metodele de bază prezentate anterior, nu poate acoperi toate cauzele care produc interblocarea în cadrul sistemelor de operare. O modalitate de rezolvare a acestei probleme, este combinarea algoritmilor de bază prezentați. Metoda propusă are la bază ideea că resursele pot fi grupate în clase şi în fiecare clasă se utilizează metoda de gestionare a resurselor cea mai adecvată. Datorită grupării resurselor în clase, o interblocare nu poate implica mai mult decât o clasă de resurse şi, cum în interiorul unei clase se aplică una dintre metodele de bază, în sistem nu pot să apară interblocări.

Exemplu. Se poate considera un sistem format din patru clase de resurse:
1. Resurse interne, adică resursele utilizate de către sistem(de exemplu, blocul de control al procesului);
2. Memoria centrală;
3. Resursele job-ului: dispozitive (discurile) şi fişiere;
4. Spațiul din memoria auxiliară alocat fiecărui job utilizator.

O metodă mixtă pentru rezolvarea interblocării în cazul acestui sistem, ordonează clasele descrise anterior, folosind în cadrul fiecărei clase următoarele abordări:
• prevenirea interblocării prin ordonarea resurselor interne deoarece, în timpul execuției, nu este necesară alegerea uneia dintre cererile nerezolvate;
• prevenirea interblocării prin achiziționare forțată a memoriei centrale, deoarece se poate evacua oricând un job în memoria auxiliară, astfel încât memoria centrală alocată acestuia să fie folosită în alt scop;
• evitarea interblocării în cazul resurselor job-ului, deoarece informațiile necesare despre formularea cererilor de resurse pot fi obținute din liniile de comandă;

• alocarea prealabilă a spațiului din memoria auxiliară asociat fiecărui job utilizator deoarece, în general, se cunoaşte necesarul maxim de memorie externă al fiecărui job.

Să ne reamintim...

Alte soluții de rezolvare a problemei interblocării se bazează pe permiterea achiziționării forțate a resurselor şi utilizarea metodelor mixte de tratare a interblocărilor.

M2.U3.9. Test de evaluare a cunoştințelor

Probleme propuse.

1.Să considerăm un sistem cu 5 procese p_1, p_2, p_3, p_4, p_5 şi trei resurse r_1, r_2, r_3, care au $10, 5$ respectiv 7 elemente. Se consideră că starea inițială este definită de:

$$A=\begin{pmatrix}0&1&0\\2&0&0\\3&0&2\\2&1&1\\0&0&2\end{pmatrix},\ M=\begin{pmatrix}7&5&3\\3&2&2\\9&0&2\\2&2&2\\4&3&3\end{pmatrix},\ N=\begin{pmatrix}7&4&3\\1&2&2\\6&0&0\\0&1&1\\4&3&1\end{pmatrix},\ D=\begin{pmatrix}2&3&0\end{pmatrix}.$$

i) Să se verifice că sistemul se află într-o stare sigură.

Indicație: Se va verifica siguranța secvenței $(p_2, p_4, p_5, p_3, p_1)$.

ii) Care va fi starea sistemului dacă procesul p_2 cere în plus 1 element al resursei r_1 şi 2 elemente ale resursei r_2. Verificați dacă starea sistemului este sigură, prin testarea siguranței secvenței $(p_2, p_4, p_5, p_1, p_3)$.

iii) Indicați alte secvențe sigure, corespunzătoare celor două stări ale sistemului.

iv) Ce se întâmplă dacă procesul p_5 face o cerere suplimentară de resurse, exprimată prin vectorul $(3, 3, 0)$.

2.Considerăm un sistem în care la un moment dat sunt în execuție procesele p_1, p_2, p_3, p_4, p_5, şi trei resurse r_1, r_2, r_3, care au 10, 2 respectiv 6 elemente. Se consideră că starea inițială este definită de:

$$A=\begin{pmatrix}0&1&0\\2&0&0\\3&0&3\\2&1&1\\0&0&2\end{pmatrix},\ C=\begin{pmatrix}0&0&0\\2&0&2\\0&0&0\\1&0&0\\0&0&2\end{pmatrix},\ D=\begin{pmatrix}2&3&0\end{pmatrix}.$$

i) Folosind algoritmul de detectare, să se verifice dacă sistemul se află în această stare sau nu.

ii) Ce se întâmplă dacă procesul p_3 face o cerere suplimentare care constă într-un element al resursei r_3.

3. Considerăm graful de alocare a resurselor prezentat în figura 2.3.6. i) Scrieți care sunt mulțimile P, R şi A. ii) Descrieți starea resurselor. iii) Descrieți starea proceselor. iv) Adăugați o cerere de resursă a unui proces astfel încât să obțineți un ciclu în graf.

4. Dați exemplu de un graf de alocare a resurselor care să conțină cicluri şi să nu genereze interblocare.

5. i) Care este graful "aşteaptă-pentru" al grafului de alocare a resurselor prezentat în figura 2. 3.7. ii) Există interblocare ?

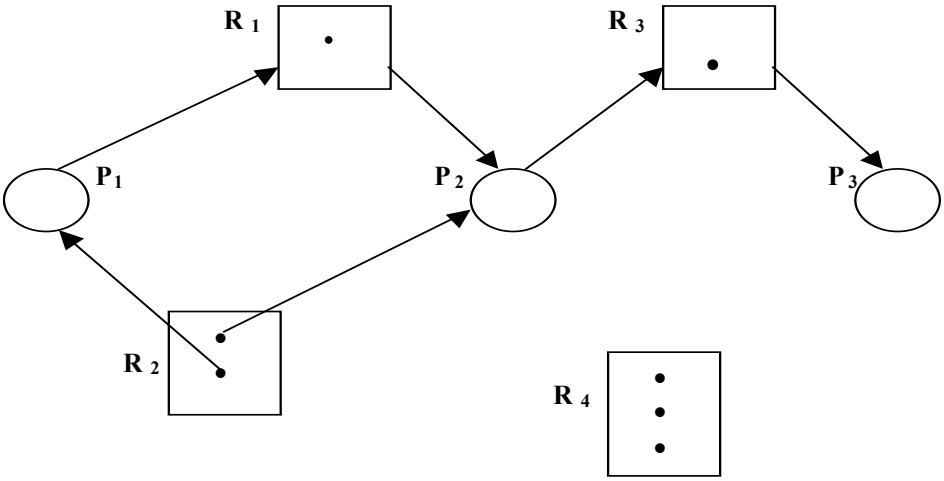


Figura 2.3.6. Graf de alocare a resurselor

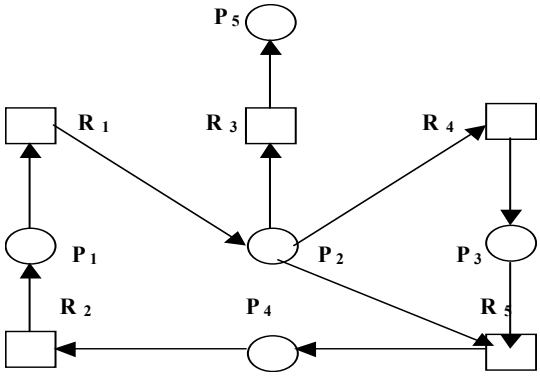


Figura 2.3.7. Graf de alocare a resurselor cu un singur element

M2.U3.10. Rezumat

În condițiile multiprogramării, procesele partajează resursele oferite de un sistem de calcul. Se spune că un set de procese se află în stare de interblocare atunci când orice proces din setul respectiv se află în aşteptarea unui eveniment de eliberare a unei resurse cerute, ce poate fi produs numai de către un proces aflat în mulțimea respectivă.

De-a lungul evoluției sistemelor de calcul, au fost propuse mai multe metode de rezolvare a problemei interblocăriiproceselor. Pentru sistemele cu prelucrare serială, o metodă potrivită este evitarea interblocării, care presupune că atunci când un proces cere resurse se verifică dacă după alocare sistemul intră într-o stare nesigură, care poate fi o stare de interblocare. Această metodă are dezavantajul încărcării cu sarcini suplimentare a CPU, de fiecare dată când se solicită noi resurse de către procese.

Calculatoarele moderne folosesc metoda detectării interblocării, care permite ca sistemul să intre în stare de interblocare şi atunci când se constată existența acesteia se aplică algoritmi de ieşire din această stare.
Alte soluții de rezolvare a problemei interblocării se bazează pe permiterea achiziționării forțate a resurselor şi utilizarea metodelor mixte de tratare a interblocărilor.

Modulul 3. Gestiunea memoriei

Cuprins

Introducere

În conformitate cu arhitectura von Neumann, memoria primară împreună cu regiştrii CPU şi memoria „cache" formează memorie executabilă, deoarece aceste componente sunt implicate în execuția unui proces. CPU poate încărca instrucțiunile acestuia numai din memoria primară. Unitățile de memorie externă (secundară) sunt utilizate pentru a stoca date pentru o mai lungă perioadă de timp. Fişierele executabile, pentru a deveni procese, precum şi informațiile prelucrate de acestea, trebuie să fie încărcate în memoria primară.

Aministratorul memoriei interne este responsabil de alocarea memoriei primare proceselor şi de a acorda asistență programatorului în încărcarea/salvarea informațiilor din/în memoria secundară. Astfel, partajrea memoriei interne de către mai multe procese şi minimizarea timpului de acces sunt obiective de bază ale administratorului memorie interne.

Componentele unui program sursă sunt nume simbolice şi formează **spațiul de nume** al programului sursă. **Faza de compilare** transformă un text sursă într-un modul obiect, adică fiecare nume simbolic este translatat într-o adresă relativă la modulul obiect. **Faza de editare de legături** grupează mai multe module, formând un fişier, numit modul absolut, stocat pe un suport extern până când se cere execuția lui. **Editorului de legături** transformă adresele din cadrul modulelor în aşa-zisele **adrese relocabile. Faza de translatare (relocare)** a adresei constă în transformarea adreselor relative la fişierul executabil, în adrese de memoria internă, realizându-se astfel **imaginea executabilă** a programului. Acest lucru este realizat de o componentă a SO, numită **încărcător**(loader). **Spațiul de adrese al unui proces** este mulțimea locațiilor alocate acestuia, atât din memoria primară, cât şi din cea secundară, servicii ale SO şi resurse. Cea mai mare parte a spațiului de adrese al unui proces se proiecteză în locații de memorie primară.

Înainte ca un program să fie executat, trebuie să-i fie alocat un spațiu din memoria primară. În condițiile multiprogramării, este posibil ca în memorie să fie prezente simultan mai multe programe. Fiecare program foloseşte zona(zonele) de memorie alocată(alocate) lui. De asemenea, pe durata execuției unui program, necesarul de memorie variază. Odată ce sistemul cunoaşte care locații de memorie urmează să fie folosite pentru execuția programului, poate să realizeze corespondența dintre adresele de memorie primară alocate procesului respectiv şi spațiul său de adrese. Deci programul executabil este translatat într-o formă finală accesibilă unității de control a CPU şi încărcat în memoria primară, la o anumită adresă de memorie. Când **contorul de program** este inițializat cu adresa primei instrucțiuni executabile(principalul punct de intrare din program), CPU începe să execute programul.

În anumite SO, administratorul memoriei poate să şteargă o parte din programul executabil din memoria primară, să-l salveze în memoria secundară şi să elibereze zona de memorie ocupată, astfel încât aceasta să poată fi alocată altor procese.

Astfel, chiar după ce un program a fost convertit într-o formă executabilă în memoria internă, el poate fi stocat şi în memoria externă, atâta timp cât este necesar. Totuşi, odată cu execuția procesului, imaginea sa din memoria internă se schimbă, fără a se modifica şi copia existentă pe disc. De exemplu, copia unei date din memoria secundară va fi modificată numai dacă procesul execută o comandă de scriere, prin care în mod explicit modifică zona de pe disc ocupată de data respectivă.

Principalele **obiective** ale gestiunii memoriei sunt: calculul de translatare a adresei(relocare); protecția memoriei; organizarea şi alocarea memoriei operative; gestiunea memoriei secundare; politici de schimb între procese, memoria operativă şi memoria secundară. Realizarea acestor funcții este un mecanism combinat soft şi hard, la care participă administratorul memoriei interne (componentă a SO) şi CPU. Odată cu evoluția componentelor hardware ale SC, s-au schimbat şi strategiile de administrare a memoriei, pentru a se valorifica aceste îmbunătățiri. Reciproc, strategiile privind gestiunea memoriei au evoluat în timp, ceea ce a condus la evoluția componentelor hardware ale sistemelor de calcul.

Competențe

La sfârşitul acestui modul, studenții vor fi capabili:

- să înțeleagă conceptele de memoria primară(internă) şi memorie executabilă;
- să înțeleagă conceptul de spațiu de adrese al unui proces;
- să identifice principalele probleme legate de administrarea memoriei;
- să explice alocarea memoriei la sistemele monoutilizator;
- să explice alocarea memoriei bazată pe partiții;
- să explice conceptul de memorie virtuală
- să explice translatarea adreselor fizice în adrese virtuale;
- să implementeze algoritmii prezentați într-un limbaj de programare cunoscut.

Unitatea de învățare M3.U1. Metode clasice de alocare a memoriei

Cuprins

M3.U1.1. Introducere

Pentru fiecare sistem de operare, specific unei generații de calculatoare, alocarea memoriei interne a constituit una dintre sarcinile principale. Conform arhitecturii von Neumman, informațiile legate de orice proces care se execută sunt încărcate în memoria internă. Dimensiunea memoriei interne este mult mai mică decât cea a spațiilor externe de memorare. De asemenea, în condițiile multiprogramării memoria internă trebuie partajată de către mai multe procese.

La generația a doua de calculatoare, prima la care apare sistemul de operare ca parte a componentei software, memoria este împărțită între sistemul de operare şi job-ul în curs de execuție. Deoarece dimensiunea memoriei este mică, apare problema faptului că dimensiunea spațiului de memorie necesar job-ului poate fi mai mare decât cel oferit de sistem. Sistemul de operarea a oferit programatorilor posibilitatea împărțirii programelor în unități logice, cărora le corespund segmente de date, care se încarcă în memorie numai atunci când este nevoie de ele.

Odată cu generația a treia de calculatoare apare multiprogramarea şi necesitatea ca spațiul de memorie internă să fie divizat în partiții şi fiecărui proces să-i fie alocată o partiție. O primă metodă este alocarea statică a memoriei, care presupune că adresele şi dimensiunile partițiilor sunt fixe. Această metodă nu foloseşte memoria eficient. Dacă dimensiunile şi adresele partițiilor se schimbă în timpul execuției, se utilizează alocarea dinamică a memoriei, care presupune o utilizare mai eficientă a acesteia.

M3.U1.2. Obiectivele unității de învățare

Această unitate de învățare îşi propune ca obiectiv principal o inițiere a studenților în studiul alocării memoriei la sistemele monoutilizator şi la cele multiutilizator bazate pe partiționarea memoriei.. La sfârşitul acestei unități de învățare studenții vor fi capabili să:

- înțeleagă şi să explice alocarea memoriei interne pentru sistemele monoutilizator;
- înțeleagă şi să explice alocarea statică a memoriei interne;
- înțeleagă şi să explice alocarea dinamică a memoriei interne;
- înțeleagă şi să explice metodele de evidență a spațiilor libere şi ocupate din memorie;
- înțeleagă şi să explice conceptele de frsgmentare a memoriei.

.

Durata medie de parcurgere a unității de învățare este de 3 ore.

M3.U1.3. Alocarea memoriri pentru sistemele monoutilizator

În cazul sistemelor cu **prelucrare în loturi** (**monoprogramare**), spațiul de memorie internă este împărțit în trei zone (figura 3.1.1):

- o zonă este alocată nucleului sistemului de operare (spațiul de adrese `0,.., a-1`);
- următoarea zonă este alocată job-ului în curs de execuție (spațiul de adrese `a`, `a+1,..,c-1`);
- zona cuprinsă între adresele `c` şi `m-1`, reprezintă un spațiu nefolosit (memoria are capacitatea de `m` locații).

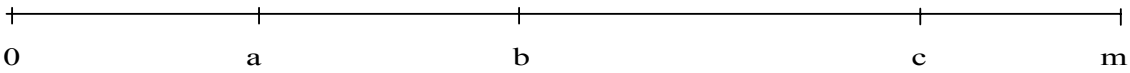


Figura 3.1.1 Alocarea memoriei la sistemele monoutilizator

La un moment dat, există un singur job în execuție, care are disponibil întreg spațiul de memorie, care începe la adresa `a`. Această metodă este specifică generației a II-a de calculatoare. Memoria internă a calculatoarelor din această generație avea o dimensiune mică. Gestiunea spațiului de adrese `a, ..,c-1` cade în sarcina utilizatorului; pentru a-şi putea rula programele de dimensiune mare, el foloseşte **tehnici de suprapunere** (overlay). Orice program este format dintr-o **parte rezidentă**, care este prezentă în memorie pe întreaga durată a execuției (adresele `a, ..,b-1`) şi o parte de suprapunere (adresele `b, ..,c-1`), în care se aduc părți ale programului (segmente), de care este nevoie la momentul respectiv. Programatorul dispune de comenzi pentru definirea segmentelor.

Exemplu: Să presupunem că avem un program care calculează suma şi produsul a două matrici precum şi norma matricilor. Partea rezidentă, va fi alocată unei funcții principale în care se fac inițializări (citirea componentelor matricilor), se memorează şi se afişează valorile obținute. Vom avea câte un modul pentru cele trei funcții carecalculează valorile indicate. La un moment dat, numai unul dintre module se poate afla în partea rezidentă.

Să ne reamintim...

În cazul sistemelor care lucrează în regim de monoprogramare**,** spațiul de memorie internă este împărțit în trei zone: o zonă este alocată nucleului sistemului de operare; următoarea zonă este alocată job-ului în curs de execuție; ultima zonă reprezintă un spațiu nefolosit.

La un moment dat, există un singur job în execuție, care are disponibil întreg spațiul de memorie.

Pentru a-şi putea rula programele de dimensiune mare utilizatorul foloseşte tehnici de suprapunere.

Înlocuiți zona punctată cu termenii corespunzători.

1. În cazul sistemelor cu prelucrare în loturi, spațiul de memorie internă este împărțit în .....**.**
2. La un moment dat, există un singur job ....., care are disponibil întreg .....**.**
3. Pentru a-şi putea rula programele de dimensiune mare, utilizatorul foloseşte .....**.**
4. Orice program este format dintr-o ....., care este prezentă în memorie pe întreaga durată a execuției şi o .....**.**

M3.U1.4. Alocarea statică a memoriei

Alocării cu partiții fixe i se mai spune şi alocare statică (MFT-Memory Fix Tasks). Este prima metodă introdusă pentru sistemele care lucrează în regim de multiprogramare. Se presupune că memoria este împărțită în N zone disjuncte, de lungime fixă numite **partiții** şi fiecare partiție este identificată cu un număr i $(i=1,..,N)$. Presupunem că dimensiunea unei partiții i este N_i şi este alocată unui proces pe toată durata execuției lui, indiferent dacă o ocupă complet sau nu. Partițiile pot avea acceaşi dimensiune sau dimensiuni diferite.

Exemplu. În figura 3.1.2. avem două tipuri de partiționări. În ambele situații, dimensiunea memoriei este de 64 de Mb şi sistemului de operare îi este alocată o partiție de 8 Mb. În cazul a) toate partițiile au aceeaşi lungime iar în cazul b) partițiile au dimensiuni diferite.

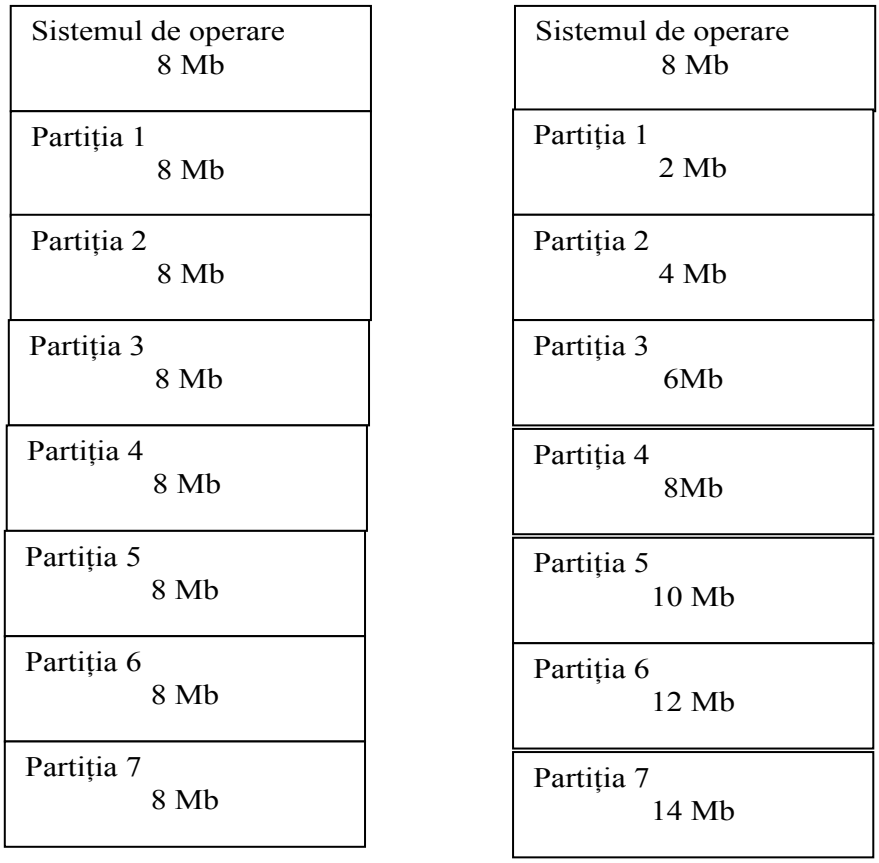


Figura 3.1.1. a) Partiții cu aceeaşi lungime b) Partiții cu lungimi diferite

Dacă partițiile au lungimi diferite, acestea sunt alocate proceselor în funcție de dimensiunile lor. Dacă un proces k are nevoie de n_k locații de memorie ele poate fi încărcat în oricare

dintre partițiile i, pentru care $N_i \geq n_k$. În timpul execuției procesului, un spațiu de memorie de dimensiune $N_i - n_k$, din partiția alocată procesului poate rămâne neutilizat. Acest fenomen se numeşte **fragmentare internă**. Problema care se pune este să se aleagă partiția astfel încât să se minimizeze diferențele de forma $N_i - n_k$.

Dacă un proces nu încape în nici una dintre partițiile existente, el nu poate fi executat. O problemă dificilă a alocării cu partiții fixe este stabilirea dimensiunii fiecărei partiții. Alegerea unor dimensiuni mari scade probabilitatea ca unele procese să nu poată fi executate, dar scade şi numărul proceselor active din sistem. În cazul în care există job-uri în sistem care aşteaptă să fie executate, dar toate partițiile libere existente la momentul respectiv sunt prea mici, apare fenomenul de **fragmentare externă a memoriei**.

Selectarea job-urilor care urmează să fie executate se face de către **planificator**, în funcție de necesarul de memorie cerut de acestea (pe baza informațiilor transmise de către utilizator sau determinate automat de către sistem) şi de partițiile disponibile la momentul respectiv; există două moduri de legare a proceselor la partiții:
• **Fiecare partiție are coadă proprie;** legarea la o anumită partiție a proceselor se va face pe baza necesității diferenței minime între dimensiunea partiției şi a procesului (**best fit**-cea mai bună potrivire).
• **O singură coadă pentru toate partițiile**; SO va alege pentru procesul care urmează să intre în lucru, în ce partiție se va executa. Selectarea lucrării se poate face prin:
- o strategie de tip FCFS(**F**irst **C**ome **F**irst **S**erved), care are dezavantajul că o anumită lucrare trebuie să aştepte în coadă chiar dacă există o partiție disponibilă în care ar încăpea iar în fața lui în coadă se află job-uri care necesită partiții de dimensiuni mai mari;
- pe baza împărțirii job-urilor în clase de priorități, în funcție de importanța lor, care poate avea dezavantajul prezentat mai sus;
- pe baza celei mai bune potrivirii între dimensiunea job-ului cu dimensiunea partiției.

Evident că metodele prezentate pot fi combinate. De exemplu, dacă avem mai multe job-uri în sistem care au aceeaşi prioritate, va fi ales cel care se potriveşte cel mai bine peste partiția care devine disponibilă. Legarea prin cozi proprii partițiilor este mai simplă din punctul de vedere al SO; în schimb, legarea cu o singură coadă este mai avantajoasă din punctul de vedere al fragmentării mai reduse a memoriei.

Alocarea cu partiții fixe a fost folosită la sistemele generației a III-a de calculatoare(IBM 360, Felix C256/512/1024), dar ea nu este recomandată pentru utilizarea în cadrul sistemelor unde nu se cunoaşte dinainte de ce spațiu de memorie are nevoie procesul pentru a fi executat, aspect întâlnit adesea în cadrul sistemelor de operare moderne.

Interschimbarea job-urilor(job-swapping) apare în cazul sistemelor cu organizarea memoriei în partiții fixe, din necesitatea ca la anumite momente unele dintre ele să fie evacuate din memorie iar altele să fie introduse în memorie. Interschimbarea joburilor necesită o memorie externă cu acces direct şi rapid, care să poată îngloba copii ale tuturor imaginilor de memorie utilizator. Toate procesele ale căror imagini de memorie se află pe disc şi care sunt gata să intre în execuție se grupează într-o coadă, în timp ce procesele existente în memorie la momentul respectiv formează altă coadă. Atunci când planificatorul doreşte să lanseze în execuție un proces, el apelează dispecerul care verifică dacă procesul se află în memorie. Dacă nu şi dacă nu există nici o partiție liberă, dispecerul evacuează din memorie unul dintre procese, introduce în locul său procesul dorit, reîncarcă registrele şi transferă controlul procesului selectat. Bineînțeles că o acțiune de acest fel presupune şi cea de salvare a contextului procesului în execuție(a conținuturilor regiştrilor utilizați de către acesta), acțiune care este destul de complexă.

Deoarece în memorie există mai multe job-uri în execuție, trebuie rezolvate două probleme: **relocarea** şi **protecția memoriei.** O soluție a rezolvării ambelor probleme este ca CPU să conțină două registre speciale: **registrul de bază** şi **registrul limită** (figura 3.1.2). Într-un fişier executabil, locațiile au adrese relative la începutul fişierului (prima locație are adresa `0`, a doua `1` ş.a.m.d.). Când lucrarea este planificată pentru execuție, în registrul de bază este încărcată adresa primei instrucțiuni din fişierul executabil, iar registrul limită va conține adresa ultimei locații din partiția respectivă. Când o locație din fişierul executabil trebuie relocată, adresa ei relativă se adaugă la adresa conținută în registrul de bază; dacă valoarea astfel obținută depăşeşte conținutul registrului limită, atunci are loc o eroare de adresare, altfel se obşine adresa unei locații din memoria internă, în care va fi încărcată locația din fişierul executabil.

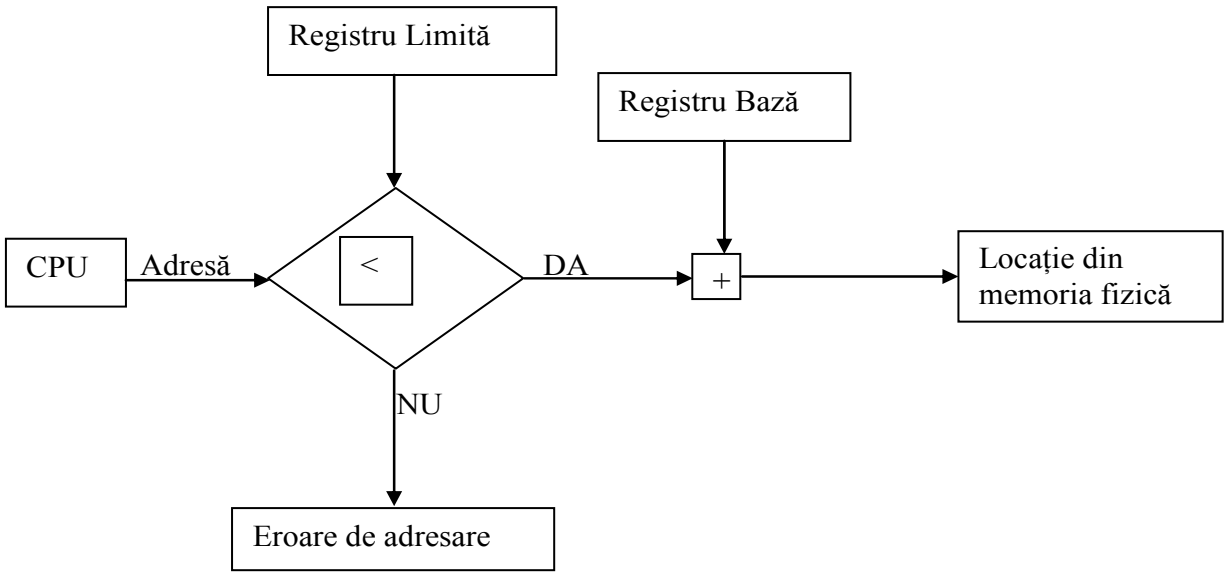


Figura 3.1.2. Relocarea adreselor cu registru limită şi registru de bază

Exemplu. Dacă se execută un job şi apare un alt job de prioritate mai înaltă, jobul de prioritate mai slabă va fi evacuat pe disc. În mod normal, un job care a fost evacuat va fi readus în aceeaşi partiție, restricție impusă atât strategia de alocare, cât şi de metoda de relocare. Dacă relocarea se face în momentul asamblării sau în momentul încărcării(relocare statică), job-ul nu poate fi transferat într-o altă partiție; dacă se foloseşte relocarea dinamică(cu registru de bază şi registru limită, de exemplu) acest lucru este posibil.

Să ne reamintim...

Alocării cu partiții fixe i se mai spune şi alocare statică . Este prima metodă introdusă pentru sistemele care lucrează în regim de multiprogramare. Se presupune că memoria este împărțită în zone disjuncte, de lungime fixă numite partiții.

În timpul execuției unui proces, dacă un spațiu de memorie din partiția respectivă rămâne neutilizat, apare fenomenul de fragmentare internă.

În cazul în care există job-uri în sistem care aşteaptă să fie executate, dar toate partițiile libere existente la momentul respectiv sunt prea mici, apare fenomenul de fragmentare externă a memoriei.

Interschimbarea job-urilor(job-swapping) apare în cazul sistemelor cu

organizarea memoriei în partiţii fixe, din necesitatea ca la anumite momente unele dintre ele să fie evacuate din memorie iar altele să fie introduse în memorie.

Relocarea şi protecţia memoriei se realizează folosind două registre speciale, registrul de bază şi registrul limită.

Înlocuiţi zona punctată cu termenii corespunzători.

1. Alocării cu partiţii fixe i se mai spune şi.....
2. Se presupune că memoria este împărţită în N zone disjuncte, de lungime fixă numite ……
3. În cazul în care există job-uri în sistem care aşteaptă să fie executate, dar toate partiţiile libere existente la momentul respectiv sunt prea mici, apare fenomenul de .....
4. În timpul execuţiei unui proces, un spaţiu de memorie din partiţia alocată procesului poate rămâne neutilizat. Acest fenomen se numeşte .....
5. Deoarece în memorie există mai multe job-uri în execuţie, trebuie rezolvate două probleme: ..... şi ..... O soluţie a rezolvării ambelor probleme este ca CPU să conţină două registre speciale: **.**..... şi .....

M3.U1.5 Alocarea dinamică a memoriei

Alocare dinamică sau alocarea cu partiţii variabile (MVT – Memory Variable Task), reprezintă o extensie a alocării cu partiţii fixe, care permite o exploatare mai eficientă a memoriei SC. În cazul multiprogramării cu partiţii fixe, problema cea mai dificilă este optimizarea dimensiunii partiţiilor, astfel încît să se minimizeze fragmentarea memoriei. De asemenea, se presupune că joburile au o dimensiune cunoscută, ipoteză care nu este în general adevărată.

Aceste inconveniente pot fi rezolvate dacă se admite **modificarea dinamică a dimensiunii partiţiilor şi adreselor lor de început şi de sfârşit**, în funcţie de solicitările adresate sistemului şi de capacitatea de memorie încă disponibilă la un moment dat. Prin folosirea acestei metode, numărul şi dimensiunea partiţiilor se modifică în timp.

În momentul în care procesul intră în sistem, el este plasat în memorie într-un spaţiu în care încape cea mai lungă ramură a sa. Spaţiul liber în care a intrat procesul, este acum descompus în două partiţii: una în care se află procesul, iar cealaltă într-un spaţiu liber care poate fi alocat altui proces. De asemenea, când un proces îşi termină execuţia, spaţiul din memorie ocupat de el este eliberat, urmând a fi utilizat de către un alt proces. Astfel, apare o alternanţă a spaţiilor libere cu cele ocupate. Pentru a se obţine spaţii libere de dimensiune cât mai mare, SO va declanşa operaţia de **alipire a unor spaţii libere vecine sau de compactare a memoriei,** adică de deplasare a partiţiilor alocate proceselor către partiţia ocupată de către nucleul SO, pentru a se concatena toate fragmentele de memorie neutilizate.

Operaţia de compactare este complexă, presupunând efectuarea de operaţii de modificare a adreselor; în practică se aleg soluţii de compromis, cum ar fi:

• Se lansează **periodic** compactarea, la un interval de timp fixat, indiferent de starea sistemului. Procesele care nu au loc în memorie aşteaptă compactarea sau terminarea altui proces.
• Se realizează o **compactare parţială** pentru a asigura loc numai procesului care asteaptă.
• Se încearcă numai mutarea unuia dintre procese, cu concatenarea spaţiilor rămase libere.

Exemplu. Presupunem că dimensiunea memoriei interne este de *64 Mo*. La început, în memorie este încărcat numai sistemul de operare (figura 3.1.3. a). Apoi se încarcă procesul *1* care solicită *10 Mo*, spațiu care i se alocă (figura 3.1.3. b) şi procesul *2* care solicită *30 Mo*, spațiu care i se alocă (figura 3.1.3. c). Procesul *1* îşi termină execuția şi eliberează spațiul ocupat. În acest moment, în memorie sunt două spații libere, de *10 Mo* respectiv de `18 Mo`. Apare un proces care cere *20 Mo*. Deoarece nu există o partiție de dimensiune suficient de mare, se face compactarea memoriei, după care procesului 3 i se poate aloca spațiul cerut.

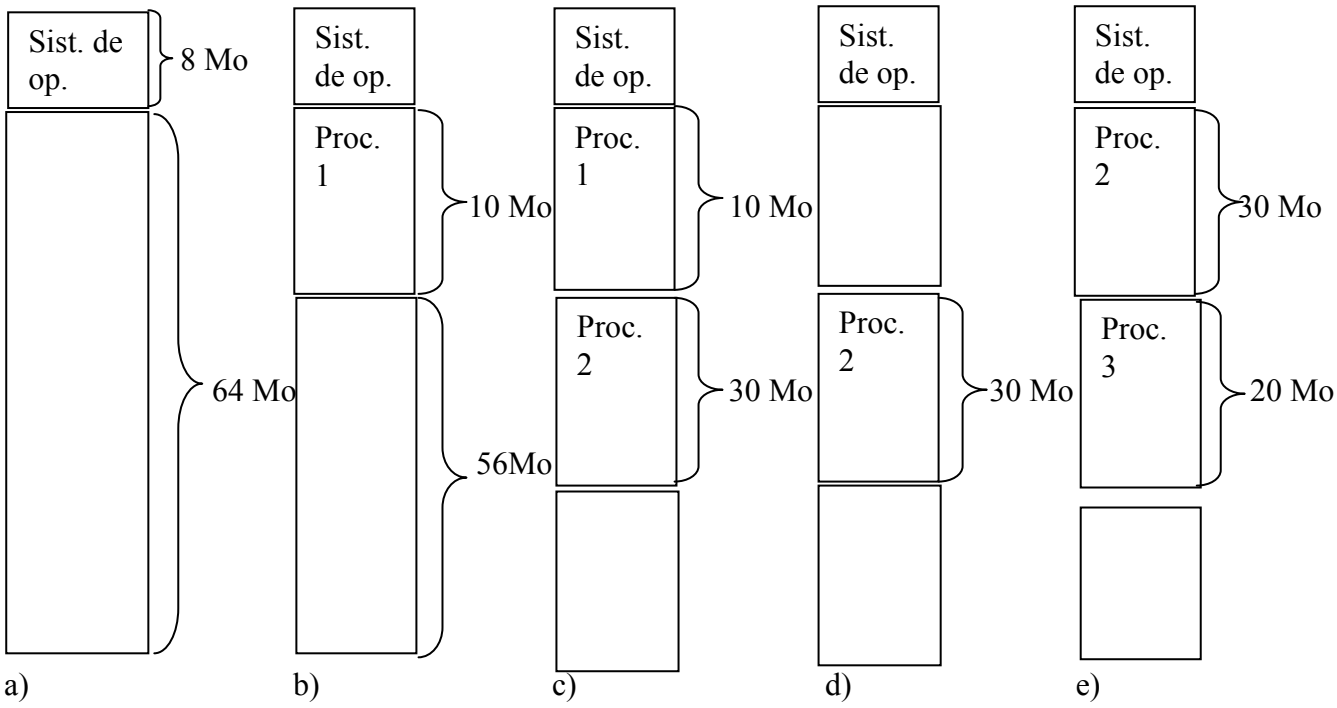


Figura 3.1.3. Exemplu de partiționare a memoriei

Administrarea spațiului din memoria internă. La un moment dat memoria se prezintă ca o alternanță a spațiilor libere cu cele ocupate. Cele libere vor fi alocate proceselor care cer memorie, iar cele ocupate, când sunt eliberate trebuie, eventual să fie concatenate cu alte spații libere, pentru a obține zone contigue de dimensiune cât mai mare, care să poată fi alocate proceselor. Deci, sunt necesare metode prin care să se țină evidența spațiilor libere şi a celor ocupate şi să se aloce spațiile de memorie solicitate. Atât spațiile libere cât şi cele ocupate pot fi considerate ca fiind nodurile a două liste dinamice. Fiecare partiție începe cu un **cuvânt de control,** care conține un **pointer** către următoarea porțiune liberă şi un câmp care conține lungimea zonei respective (figura 3.1.4). La fel se întâmplă în cazul unei **zone ocupate.** Administratorul memoriei interne trebuie să rezolve următoarele probleme:
- După eliberarea unui spațiu de către un proces care şi-a terminat execuția, se pot obține două spații de memorie libere adiacente; este necesară **concatenarea** lor, pentru a se obține un spațiu suficient de mare care poate fi alocat unui proces.

- Căutarea unui spațiu liber care să poată fi alocat unui proces, se realizează prin parcurgerea listei, utilizând informațiile din cuvintele de control ale partițiilor.

Ppartiția care va fi alocată procesului se selectează după următoarele strategii:
• **Metoda primei potriviri (First-Fit):** partiția solicitată este alocată în prima zonă liberă în care începe procesul. Principalul avantaj al metodei este simplitatea căutării de spațiu liber.

• **Metoda celei mai bune potriviri (Best-Fit):** căutarea acelei zone libere care lasă după alocare cel mai puțin spațiu liber. **Avantajul** metodei constă în economisirea zonelor de memorie mai mari. **Dezavantajul** este legat de timpul suplimentar de căutare şi generarea blocurilor de lungime mică, adică **fragmentarea internă excesivă**. Primul neajuns este eliminat parțial, dacă lista de spații libere se păstrează nu în ordinea crescătoare a adreselor, ci în ordinea crescătoare a lungimilor spațiilor libere, dar algoritmul se complică.
• **Metoda celei mai rele potriviri (Worst-fit)** este dual metodei Best-Fit; se caută acele zone libere care lasă după alocare cel mai mult spațiu liber. După alocare rămâne un spațiu liber mare, în care poate fi plasată o altă partiție.
- divizarea unei partiții libere în două spații, unul care poate fi alocat unui proces (introdus în lista spațiilor ocupate) şi unul care să rămână liber.

Toate aceste operații se pot realiza prin folosirea operațiilor cu pointeri şi prelucrarea cuvântului de control.

Să ne reamintim...
Alocare dinamică sau alocarea cu partiții variabile reprezintă o extensie a alocării cu partiții fixe, care permite o exploatare mai eficientă a memoriei SC. Această metodă permite modificarea dinamică a dimensiunii partițiilor, în funcție de solicitările adresate sistemului şi de capacitatea de memorie disponibilă la un moment dat. Prin folosirea acestei metode, se optimizează dimensiunile partițiilor, astfel încît să se minimizeze fragmentarea memoriei.

Pentru a se obține spații libere de dimensiune cât mai mare, SO va declanşa operația de alipire a unor spații libere vecine sau de compactare a memoriei**,** adică de deplasare a partițiilor active către partiția ocupată de către nucleul SO, pentru a se concatena toate fragmentele de memorie neutilizate.

Evidența spațiilor libere şi a celor ocupate din memoria internă se face folosind liste înlănțuite.

Înlocuiți zona punctată cu termenii corespunzători.
1. Alocare dinamică sau alocarea cu ..... , reprezintă o extensie a alocării cu partiții fixe, care permite o exploatare ..... a memoriei SC. Prin folosirea acestei metode, ... se modifică în timp.
2. Atât spațiile libere cât şi cele ocupate pot fi considerate ca fiind ..... a două liste dinamice. Fiecare zonă liberă începe cu un ....., care conține un ..... către următoarea porțiune liberă şi un câmp care conține ..... zonei respective
3. După eliberarea unui spațiu de către un proces care şi-a terminat execuția, se pot obține două ..... libere adiacente; este necesară ..... lor, pentru a se obține un ..... de mare care poate fi .....unui proces.
4. Prin metoda primei potriviri (First-Fit) ..... este alocată în prima zonă liberă în care încape. Principalul avantaj al metodei este ..... de spațiu liber.
5. Prin metoda celei mai bune potriviri (Best-Fit) se caută acea ..... care lasă după alocare ..... Avantajul metodei constă în .....zonelor de memorie mai mari. Dezavantajul este legat de ..... şi generarea blocurilor de lungime mică, adică .....
6. Metoda celei mai rele potriviri (Worst-fit) este ..... metodei Best-Fit; se caută acele zone libere care lasă după alocare ..... După alocare rămâne un spațiu liber mare, în care poate fi plasată .....

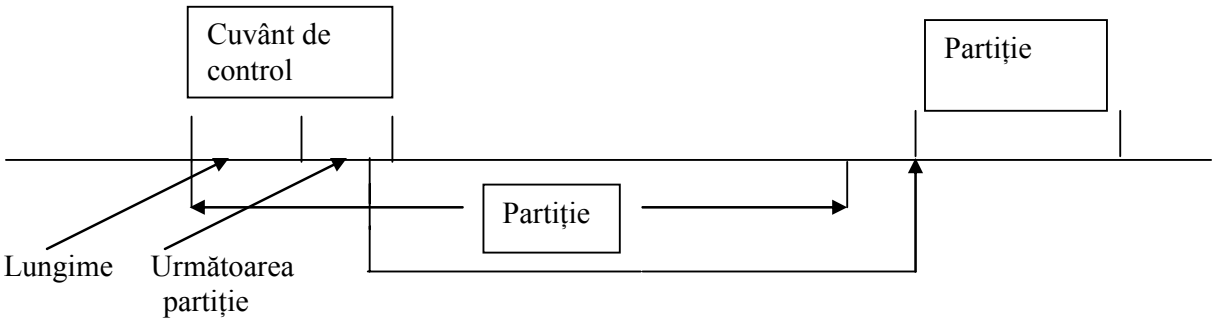


Figura 3.1.4. Înlănțuirea a două partiții

M3.U1.6. Metoda alocării memoriei prin camarazi

Cele două metode de alocare a memoriei interne prezentate au anumite deficiențe. Alocarea cu partiții fixe generează fenomenele de fragmentare a memoriei iar alocarea dinamică presupune o schemă relativ complicată de evidență şi compactare a spațiilor libere şi ocupate.

Metoda alocării prin camarazi (Buddy-system) se bazează pe reprezentarea binară a adreselor şi faptul că dimensiunea memoriei interne este un multiplu al unei puteri a lui 2. Presupunem că se dimensiunea memoriei interne este de forma $cx2^n$, iar unitatea de alocare a memoriei este de forma 2^m.

Exemplu. Dacă sistemul are o memorie internă de 32 *Mo*, atunci $c=1$ şi $n=25$. Dacă dimensiunea memoriei interne este de 192 *Mo*, $c=3$ şi $n=26$. Dacă dimensiunea memoriei este de 1*Go*, atunci $c=1$ şi $n=30$. De asemenea, se poate considera că unitatea de alocare este de 256 *Ko*, adică 2^8.

Ținând cont de proprietățile operațiilor cu puteri ale lui 2, atât dimensiunile spațiilor alocate, cât şi ale celor libere sunt de forma 2^k, cu $k \le m \le n$; sistemul va păstra liste separate ale adreselor spațiilor disponibile, în funcție de dimensiunea lor exprimată ca putere a lui 2. Vom numi lista de ordin k, lista tuturor adreselor unde încep spații libere de dimensiune 2^k. Vor exista astfel $n-m+1$ liste de spații disponibile.

Exemplu. Dacă considerăm că dimensiunea memoriei interne este de 192 *Mo*, vom avea 17 liste: lista de ordin 8, având dimensiunea unui spațiu de 256 octeți; lista de ordin 9, cu spații de dimensiune 512 ş.a.m.d.

Presupunem că, fiecare spațiu liber(ocupat) de dimensiune 2^k, are adresa de început un multiplu de 2^k. Două spații libere se numesc **camarazi de ordinul** k, dacă adresele lor A_1 şi A_2 verifică una dintre proprietățile următoare:

$$A_1<A_2 \ , \ A_2=A_1+2^k \text{ şi } A_1 \mod 2^{k+1}=0$$
$$A_2<A_1, \ A_1=A_2+2^k \text{ şi } A_2 \mod 2^{k+1}=0$$

Observație. Atunci când într-o listă de ordinul k apar doi camarazi, sistemul îi concatenează într-un spațiu de dimensiune 2^{k+1} şi reciproc, un spațiu de dimensiune 2^{k+1} se poate împărți în două spații de dimensiune 2^k.

Algoritmul de alocare de memorie.
Pas 1. Fie o numărul de octeți solicitați. Se determină $min\{p/\ m \le p \le n,\ o \le 2^p\}$.
Pas 2. Se determină $k=min\{i/p \le i \le n$ şi *lista de ordin i este nevidă*$\}$.

Pas 3. Dacă $k=p$, atunci aceasta este alocată şi se şterge din lista de ordinul p altfel se alocă primii 2^p octeţi, se şterge zona din lista de ordinul k şi se creează în schimb alte $k-p$ zone libere, având dimensiunile $2^p, 2^{p+1}, \ldots, 2^{k-1}$.

Observaţie. Pasul 3 al algoritmului se bazează pe egalitatea

$$2^k - 2^p = 2^p + 2^{p+1} + \ldots + 2^{k-1}.$$

Exemplu. Se doreşte alocarea a 1000 octeţi, deci $p=10$. Nu s-au găsit zone libere de dimensiune 2^{10} sau 2^{11} sau 2^{12}. Prima zonă liberă de dimensiune 2^{13} are adresa de început $5x2^{13}$ şi o notăm cu I. Ca rezultat al alocării a fost ocupată zona A de dimensiune 2^{10} şi au fost create încă trei zone libere: B de dimensiune 2^{10}, C de dimensiune 2^{11} şi D de dimensioune 2^{12}. Zonele B, C şi D se trec respectiv în listele de ordine 10, 11 şi 12, iar zona I se şterge din lista de ordin 13.

Algoritmul de eliberare.

Pas 1. Fie 2^p dimensiunea zonei eliberate. Se introduce zona respectivă în lista de ordinul p.

Pas 2. $k=p$

Pas 3. Verifică dacă există camarazi de ordin k:

Dacă **da**, efectuează **comasarea** lor; **Şterge** cei doi camarazi; **Introdu** noua zonă liberă de dimensiune 2^{k+1} în lista de ordin $k+1$.

Pas 4. $k=k+1$; goto Pas 3.

Exemplu.

Să presupunem că la un moment dat zonele A , C şi D de adrese $5x2^{13}$, respectiv $21x2^{11}$ şi $11x2^{12}$ sunt libere, iar zona B de adresa $41x2^{11}$ este ocupată (figura 3.1.4). Avem: $41*2^{10}-5*2^{13}=2^{10}$; $21*2^{11}-41*2^{10}=2^{10}$; $11*2^{12}-21*2^{11}=2^{11}$.

Se observă că zonele sunt adiacente, în ordinea A, B, C, D. În conformitate cu paşii descrişi mai sus, se execută următoarele acţiuni:

- Se trece zona B în lista de ordin 10.
- Se observă că zonele A şi B sunt camarazi; cele două zone sunt comasate şi formează o nouă zonă X. Zona X se trece în lista de ordin 11, iar zonele A şi B se şterg din lista de ordin 10.
- Se observă că zonele X şi C sunt camarazi; ele sunt comasate şi formează o zonă Z care se trece în lista de ordin 12, înlocuind zonele X şi C din lista de ordin 11.
- Se observă că Z şi D sunt camarazi; ele sunt şterse din lista de ordin 12, iar în lista de ordin 13 se introduce rezultatul comasării lor.

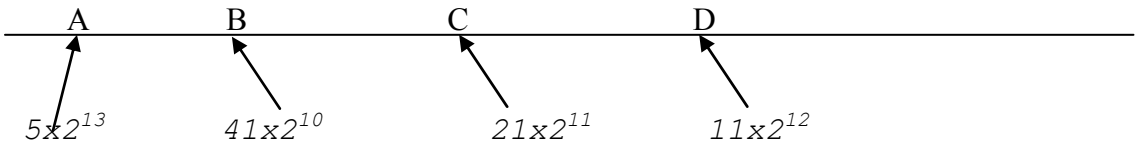


Figura 3.1.4. Exemplu de camarazi

Să ne reamintim...

Metoda alocării prin camarazi (Buddy-system) se bazează pe reprezentarea binară a adreselor şi faptul că dimensiunea memoriei interne este un multiplu al unei puteri a lui 2.
Sistemul păstrează liste separate ale adreselor spațiilor disponibile, în funcție de dimensiunea lor exprimată ca putere a lui 2. Vom numi lista de ordin k, lista tuturor adreselor unde încep spații libere de dimensiune 2^k.

Înlocuiți zona punctată cu termenii corespunzători.
1. Metoda alocării prin camarazi (Buddy-system) se bazează pe ..... a adreselor şi faptul că dimensiunea memoriei interne este un ..... a lui 2.
2. Două spații libere se numesc **camarazi de ordinul** k, dacă adresele lor A_1 şi A_2 verifică una dintre proprietățile următoare:

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

3. Atunci când într-o listă de ordinul k apar doi camarazi, sistemul îi concatenează într-un spațiu de dimensiune 2^{k+1} şi reciproc, un spațiu de dimensiune 2^{k+1} se poate împărți în două spații de dimensiune 2^k.

M3.U1.7. Test de evaluare a cunoştințelor
I. Selectați varianta corectă.

1. Sistemele pentru care la un moment dat, există un singur job în execuție, care are disponibil întreg spațiul de memorie, este specific:

a) generației a I-a de calculatoare. ☐ b) generației a II-a de calculatoare. ☐

c)generației a III-a de calculatoare. ☐ d) generației a IV-a de calculatoare. ☐

2. Fragmentarea memoriei apare:

a) În cazul alocării cu partiții. ☐ b) În cazul alocării paginate. ☐

c) În cazul alocării de memorie "cache". ☐ d) În cazul alocării de memorie secundară. ☐

3. Fragmentare internă înseamnă:

a) Un fişier executabil este memorat pe mai multe discuri care nu sunt ocupate complet. ☐ b)Un fişier executabil este memorat pe mai multe blocuri fizice care nu sunt ocupate complet. ☐

c) Rămâne spațiu liber pe disc alocat unui fişier executabil neutilizat în cursul execuției acestuia. ☐ d) Rămâne spațiu liber de memorie internă alocat unui proces neutilizat în cursul execuției acestuia. ☐

4.Fragmentare externă este specifică:

a) Alocării dinamice a memoriei interne proceselor. ☐ c) Alocării dinamice a memoriei interne fişierelor executabile. ☐

b) Alocării statice a memoriei interne proceselor. ☐ d) Alocării statice a memoriei interne fişierelor executabile. ☐

5.Fragmentare externă însemnă:

a) Unui proces îi sunt alocate mai multe partiţii din memoria internă pe care nu le foloseşte în totalitate. ☐

b) Unui fişier executabil îi sunt alocate mai multe blocuri fizice din memoria externă pe care nu le foloseşte în totalitate. ☐

c) În memoria internă există partiţii libere a căror dimensiune este insuficientă pentru ca un proces să fie executat. ☐

d) În memoria externă există partiţii libere a căror dimensiune este insuficientă pentru ca un proces să fie executat. ☐

6. Compactarea memoriei înseamnă:

a) Deplasarea partiţiilor alocate proceselor către partiţia ocupată de către nucleul SO. ☐

b) Deplasarea partiţiilor alocate proceselor către partiţia ocupată de către compilatorul SO. ☐

c) Deplasarea partiţiilor alocate proceselor către locaţia de memorie cu valoarea cea mai mare. ☐

d) Deplasarea partiţiilor alocate proceselor către locaţia de memorie cu valoarea cea mai mică. ☐

7. Fiecare partiţie începe cu un cuvânt de control**,** care conţine:

a) Un pointer către următoarea partiţie din listă şi un câmp care conţine lungimea fişierului executabil care va fi încărcat în partiţia respectivă. ☐

b) Un pointer către următoarea partiţie din listă şi un câmp care conţine lungimea zonei respective. ☐

c) Un pointer către fişierul executabil care va fi încărcat în partiţie şi un câmp care conţine lungimea partiţiei. ☐

d) Un pointer către fişierul executabil care va fi încărcat în partiţie şi un câmp care conţine lungimea fişierului respectiv. ☐

8**.** Care strategie de alocare a unei partiţii unui proces, poate lăsa un spaţiu suficient de mare care poate fi o partiţie pentru un alt proces:

a) Metoda primei potriviri. ☐

b) Metoda celei mai bune potriviri. ☐

c) Metoda celei mai rele potriviri. ☐

d) Metoda potrivirii optime. ☐

II. Problemă propusă.

Dacă într-un sistem partiţiile au dimensiunile de 150 Ko, 520 Ko, 230 Ko, 350 Ko şi 620 Ko şi poziţionarea lor în memorie este în această ordine şi se pune problema încărcării în memorie a 4 procese a căror dimensiune este de 138 Ko, 477 Ko, 192 Ko şi 396 Ko să se specifice partiţia în care va fi plasat fiecare proces, considerând cele trei metode (“first-fit”, “best-fit”, “worst-fit”) de alocare. Comentaţi eficienţa alocării.

M3.U1.8. Rezumat

În cazul sistemelor care lucrează în regim de monoprogramare**,** spațiul de memorie internă este împărțit în trei zone: o zonă este alocată nucleului sistemului de operare; următoarea zonă este alocată job-ului în curs de execuție; ultima zonă reprezintă un spațiu nefolosit. La un moment dat, există un singur job în execuție, care are disponibil întreg spațiul de memorie. Pentru a-şi putea rula programele de dimensiune mare utilizatorul foloseşte tehnici de suprapunere.

Alocării cu partiții fixe i se mai spune şi alocare statică. Este prima metodă introdusă pentru sistemele care lucrează în regim de multiprogramare. Se presupune că memoria este împărțită în zone disjuncte, de lungime fixă numite partiții. În timpul execuției unui proces, dacă un spațiu de memorie din partiția respectivă rămâne neutilizat, apare fenomenul de fragmentare internă.

În cazul în care există job-uri în sistem care aşteaptă să fie executate, dar toate partițiile libere existente la momentul respectiv sunt prea mici, apare fenomenul de fragmentare externă a memoriei.

Interschimbarea job-urilor(job-swapping) apare în cazul sistemelor cu organizarea memoriei în partiții fixe, din necesitatea ca la anumite momente unele dintre ele să fie evacuate din memorie iar altele să fie introduse în memorie.

Alocarea dinamică sau alocarea cu partiții variabile reprezintă o extensie a alocării cu partiții fixe, care permite o exploatare mai eficientă a memoriei. Această metodă permite modificarea dinamică a dimensiunii partițiilor, în funcție de solicitările adresate sistemului şi de capacitatea de memorie disponibilă la un moment dat. Prin folosirea acestei metode, se optimizează dimensiunile partițiilor, astfel încît să se minimizeze fragmentarea memoriei.

Pentru a se obține spații libere de dimensiune cât mai mare, sistemul de operare va declanşa operația de alipire a unor spații libere vecine sau de compactare a memoriei**,** adică de deplasare a partițiilor active către partiția ocupată de către nucleul sistemului de operare, pentru a se concatena toate fragmentele de memorie neutilizate. Evidența spațiilor libere şi a celor ocupate din memoria internă se face folosind liste înlănțuite.

Unitatea de învățare M3.U2. Segmentarea şi paginarea memoriei

Cuprins

M3.U2.1. Introducere

Chiar şi calculatoarele moderne au o dimensiune a memoriei interne relativ mică, dacă o raportăm la spațiile de memorie externă. În condițiile multiprogramării, sistemul de operare trebuie să satisfacă cerințele de memori ale mai multor procese. În acest sens, memoria internă internă este o resursă care trebuie gest

Calculatoarele moderne folosesc conceptul de memorie virtuală, ce reprezintă extinderea spațiului de adrese al memoriei fizice şi care se sprijină pe ideea asocierii de adrese fizice obiectelor programului în timpul execuției lui. Compilatorul şi link-editorul creează un modul absolut, căruia încărcătorul îi asociază adrese fizice înainte ca programul să fie executat. Facilitățile hardware permit administratorului memoriei să încarce automat porțiuni ale spațiului de adrese virtuale în memoria primară, în timp ce restul spațiului de adrese este păstrat în memoria secundară.

Sistemele cu paginare transferă blocuri de informație de dimensiune fixă între memoria secundară şi cea primară. Datorită dimensiunii fixe a paginii virtuale şi a celei fizice, translatarea unei adrese virtuale într-una fizică devine o problemă reletiv simplă, prin mecanismele tabelei de pagini şi memoriei virtuale.Politicile de încărcare şi de înlocuire a paginilor, permit ca paginile să fie încărcate în memorie, numai atunci când se face o referință la o locație de memorie pe care o conțin. Politicile de plasare şi de înlocuire a paginilor, definesc reguli prin care se optimizează transferul de informații din memoria virtuală în memoria primară şi reciproc.

Segmentarea este o alternativă la paginare. Ea diferă de paginare prin faptul că unitățile de transfer dintre memoria secundară şi cea primară variază în timp. Dimensiunea segmentelor trebuie să fie în mod explicit definită de către programator sau sistem. Translatarea unei adrese virtuale segmentate într-o adresă fizică este mult mai complexă decât translatarea unei adrese virtuale paginată. Segmentarea se sprijină şi pe sistemul de fişiere, deoarece segmentele sunt stocate pe memoriile externe sub formă de fişiere. Segmentarea cu paginare valorifică avantajele oferite de cele două metode, fiind utilizată de sistemele de calcul moderne.

Memoriile cu acces rapid cresc performanțele unui sistem de calcul. Metodele de proiecție a spațiului din memoria internă în memoria cache, permite ca orice locație fizică să fie accesată mult mai rapid.

M3.U2.2. Obiectivele unității de învățare

Această unitate de învățare își propune ca obiectiv principal o introducere a studenților în problematica sistemelor de operare. La sfârșitul acestei unități de învățare studenții vor fi capabili să:

- înțeleagă și să explice conceptul de memorie virtuală;
- înțeleagă și să explice translatarea adreselor virtuale în adrese fizice;
- înțeleagă și să explice alo;
- înțeleagă și să explice tipurile de sisteme de operare;
- înțeleagă și să explice încărcarea sistemului de operare.

Durata medie de parcurgere a unității de învățare este de 3 ore.

M3.U2.3. Memoria virtuală

Dacă presupunem că `n` este dimensiunea magistralei de adrese, atunci numărul maxim de locații adresabile este 2^n . Dimensiunea memoriei interne este mult mai mică decât această valoarea. Acest fapt sugerează prelungirea spațiului de adrese pe un disc rapid. Mulțimea acestor locații de memorie formează memoria virtuală.

Utilizarea memoriei virtuale este o tehnică ce permite execuția proceselor fără a impune necesitatea ca întreg fișierul executabil să fie încărcat în memorie și capacitatea de a adresa un spațiu de memorie mai mare decât cel din memoria internă a unui SC.

Spațiul de adrese al procesului este divizat în părți care pot fi încărcate în memoria internă atunci când execuția procesului necesită acest lucru și transferate înapoi în memoria secundară, când nu mai este nevoie de ele. Spațiul de adrese al unui program se împarte în **partea de cod**, cea **de date** și cea **de stivă**, identificate atât de compilator cât și de mecanismul hardware utilizat pentru relocare. Partea (segmentul) de cod are un număr de componente mai mare, fiind determinată de fazele de execuție (logica) ale programului.

Exemplu: Aproape toate programele conțin o fază necesară inițializării structurilor de date utilizate în program, alta pentru citirea datelor de intrare, una (sau mai multe) pentru efectuarea unor calcule, altele pentru descoperirea erorilor și una pentru ieșiri. Analog, există partiții ale segmentului de date.

Această caracteristică a programului se numește localizare a **referințelor în spațiu** și este foarte importantă în strategiile utilizate de către sistemele de memorie virtuală. Când o anumită parte a programului este executată, este utilizată o anumită porțiune din spațiul său de adrese, adică este realizată o localizare a referințelor. Când se trece la o altă fază a calculului, corespunzătoare logicii programului, este referențiată o altă parte a spațiului de adrese și se schimbă această localizare a referințelor.

Exemplu: În figura 3.2.1, spațiul de adrese este divizat în *5* părți. Numai părțile *1* și *4* din spațiul de adrese corespund unor faze ale programului care se execută la momentul respectiv, deci numai acestea vor fi încărcate în memoria internă. Părți diferite ale programului vor fi încărcate în memoria primară la momente diferite, în funcție de localizarea în cadrul procesului.

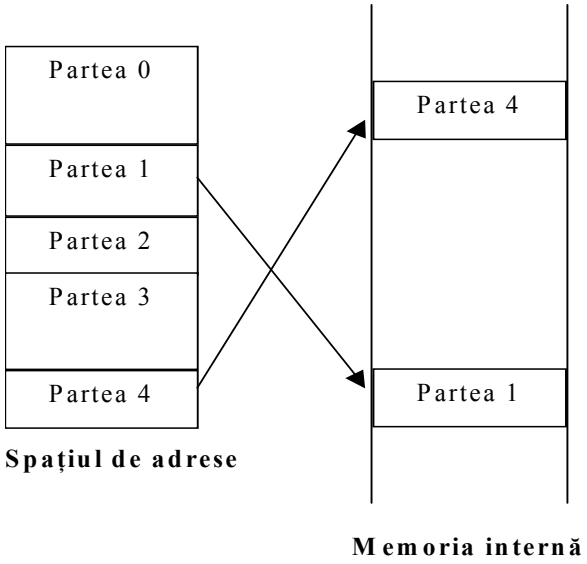


Figura 3.2.1 Memoria fizică şi cea virtuală

Sarcina administratorului memoriei este de a deduce localizarea programului şi de a urmări încărcarea în memorie a partiţiilor din spaţiul de adrese corespunzătoare, precum şi de a ţine evidenţa acestora în memoria internă, atâta timp cât sunt utilizate de către proces.

Administratorul memoriei virtuale alocă porţiuni din memoria internă, care au aceeaşi dimensiune cu cele ale partiţiilor din spaţiul de adrese şi încarcă imaginea executabilă a părţii corespondente din spaţiul de adrese într-o zonă din memoria primară. Acest lucru are ca efect utilizarea de către proces a unei cantităţi de memorie mult mai reduse.

Translatarea adreselor virtuale. Funcţia de translatare a adreselor virtuale, notată cu Ψ_t este o corespondenţă variabilă în timp a **spaţiului de adrese virtuale** ale unui proces, în **spaţiul de adrese fizice**, adică mulţimea tuturor locaţiilor din memoria internă alocate acelui proces. Se cunosc două metode de virtualizare: **alocare paginată şi alocare segmentată.** Deci:

$$\Psi_t :< Spatiul\_de\_adrese\_relative > \rightarrow < Spatiul\_de\_adrese\_fizice > \cup \{\Omega\}$$

în care: t este un număr întreg, care reprezintă timpul virtual al procesului iar Ω este un simbol care corespunde adresei nule. Când un element i al spaţiului de adrese virtuale este încărcat în memoria internă, $\Psi_t(i)$ este adresa fizică unde adresa virtuală i este încărcată.

Dacă la momentul virtual t, $\Psi_t(i) = \Omega$ şi procesul face referinţă la locaţia i, atunci sistemul întreprinde următoarele acţiuni:

1. Administratorul memoriei cere oprirea execuţiei procesului.
2. Informaţia referenţiată este regăsită în memoria secundară şi încărcată într-o locaţie de memorie k.
3. Administratorul memoriei schimbă valoarea funcţiei Ψ, $\Psi_t(i) = k$.
4. Administratorul memoriei cere reluarea execuţiei programului.

Observăm că locaţia referenţiată din spaţiul de adrese virtuale nu este încărcată în memoria primară, după ce s-a declanşat execuţia instrucţiunii cod maşină respective. Acest lucru va declanşa **reexecutarea** instrucţiunii după ce locaţia respectivă a fost încărcată din memoria virtuală în memoria primară. De asemenea, dimensiunea spaţiului de adrese virtuale ale unui

proces, este **mai mare** decât dimensiunea spațiului de adrese fizice alocat procesului, spre deosebire de metodele când întregul fișier executabil era încărcat în memorie.

Să ne reamintim...

Dimensiunea memoriei interne este mult mai mică decât numărul maxim de locații adresabile această valoarea. Acest fapt sugerează prelungirea spațiului de adrese pe un disc rapid. Mulțimea acestor locații de memorie formează memoria virtuală.

Spațiul de adrese al unui program se împarte în **partea de cod**, cea **de date** şi cea **de stivă**, identificate atât de compilator cât şi de mecanismul hardware utilizat pentru relocare. Când o anumită parte a programului este executată, este utilizată o anumită porțiune din spațiul său de adrese, adică este realizată o localizare a referințelor.

Înlocuiți zona punctată cu termenii corespunzători.

1. Dacă presupunem că `n` este dimensiunea ....., atunci numărul maxim de locații adresabile este .....
2. Spațiul de adrese al unui program se împarte în **.....**, cea **.....** şi cea **.....**, identificate atât de compilator cât şi de mecanismul hardware utilizat pentru relocare.
3. Când o anumită parte a programului este executată, este utilizată o anumită porțiune din ....., adică este realizată o .....
4. Funcția de translatare a adreselor virtuale este o corespondență variabilă în timp a ..... ale unui proces, în .....
5. Dimensiunea spațiului de adrese virtuale ale unui proces este ..... decât dimensiunea spațiului de adrese fizice alocat procesului

M3.U2.4. Alocarea paginată a memoriei

Paginarea memoriei a apărut la diverse SC pentru a evita fragmentarea memoriei interne, care apare la metodele anterioare de alocare. Memoria virtuală este împărțită în zone de lungime fixă numite **pagini virtuale.** Paginile virtuale se păstrează în memoria secundară. Memoria operativă este împărțită în zone de lungime fixă, numite **pagini fizice.** Paginile virtuale şi cele fizice au aceeaşi lungime, lungime care este o putere a lui *2* şi care este o constantă a sistemului.

Exemplu: Lungimea poate fi `1Ko`, `2Ko` etc.

Să presupunem că 2^k este dimensiunea unei pagini; daca `n` este adresa unei locații din memoria fizică, atunci conform teoremei de împărțire cu rest, $n=f2^k+d$; `f` este numărul de pagină în care se află locația de adresă *n* iar *d* este adresa (deplasamentul) din cadrul paginii; $f2^k$ este adresa din memoria primară a primei locații din pagina fizică. Perechea `(f,d)` se numeşte adresa paginată fizică a locației de memorie. Analog se definește notiunea de adresă paginată a unei locații din memoria virtuală, pe care o vom nota cu `(p,d)`. Deoarece fiecare pagină are aceeaşi dimensiune 2^k, adresa virtuală `i` poate fi convertită într-un număr de pagină şi un deplasament în cadrul paginii, numărul de pagină fiind `i div` 2^k, iar deplasamentul `i mod` 2^k.

Orice pagină virtuală din spațiul de adrese al procesului, poate fi încărcată în oricare din paginile fizice din memoria internă alocate acestuia. Acest lucru este realizat printr-o componentă hardware numită MMU(**M**emory **M**anagement **U**nit), care implementează funcția Ψ_t (figura 3.2.2).

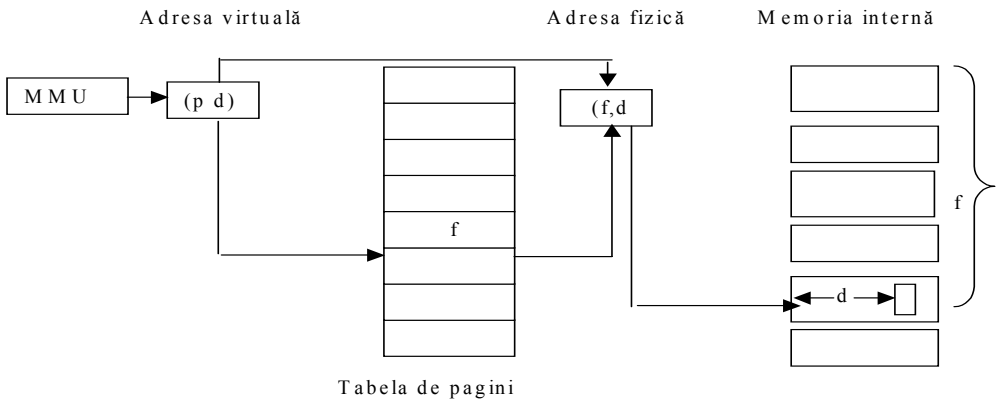


Figura 3.2.2. Translatarea unei pagini virtuale într-una fizică

Dacă este referențiată o locație dintr-o pagină care nu este încărcată în memoria internă, MMU opreşte activitatea CPU, pentru ca SO să poată executa următorii paşi:

1. Procesul care cere o pagină neîncărcată în memoria internă este suspendat.
2. Administratorul memoriei localizează pagina respectivă în memoria secundară.
3. Pagina este încărcată în memoria internă, eventual în locul altei pagini, dacă în memoria internă nu mai există pagini fizice libere alocate procesului respectiv şi în acest caz, tabela de pagini este modificată.
4. Execuția procesului se reia din locul în care a fost suspendat.

Fiecare proces are propria lui tabelă de pagini, în care este trecută adresa fizică a paginii virtuale, dacă ea este prezentă în memoria operativă. La încărcarea unei noi pagini virtuale, aceasta se depune într-o pagină fizică liberă. Deci, în memoria operativă, paginile fizice sunt distribuite în general necontinuu, între mai multe procese. Spunem că are loc o **proiectare a spatiului virtual peste cel real.** Acest mecanism are avantajul că foloseşte mai eficient memoria operativă, fiecare program ocupând numai memoria strict necesară la un moment dat. Un alt avantaj este posibilitatea folosirii în comun, de către mai multe programe, a instrucțiunilor unor proceduri. O procedură care permite acest lucru se numeste **procedură reentrantă.**

Evidența paginilor virtuale încărcate în pagini fizice se poate realiza prin tabela de pagini, încărcată în memorie.Tabela de pagini poate fi privită ca o funcție, care are ca argument numărul de pagină virtuală şi care determină numărul de pagină fizică. Pe baza deplasamentului se determină locația din memoria fizică unde se va încărca locația virtuală referențiată. Această metodă ridică două mari probleme:

1.Tabela de pagini poate avea un număr mare de intrări. Calculatoarele moderne folosesc adrese virtuale pe cel puțin `32` de biți.

Exemplu: Dacă o pagină are dimensiunea de `4K`, atunci numărul intrărilor în tabelă este mai mare decât `1000000`. În cazul adreselor pe 64 de biți numărul intrărilor în tabelă va fi mult mai mare.

2. Corespondența dintre locația de memorie virtuală şi cea din memoria fizică trebuie să se realizeze cât mai rapid posibil; la un moment dat, o instrucțiune cod maşină poate referenția una sau chiar două locații din memoria virtuală.

O metodă mult mai eficientă, care înlocuieşte căutarea secvențială cu cea arborescentă este **organizarea tabelei pe mai multe niveluri.** Astfel, o adresă virtuală pe 32 de biți este un triplet (Pt_1, Pt_2, d), primele două câmpuri având o lungime de 10 biți, iar ultimul de 12 biți. În figura 3.2.3 este ilustrată o astfel de tabelă de pagini. Observăm că tabela este organizată pe două niveluri. Numărul de intrări în tabela de nivel 1 (corespunzătoare lui Pt_1) este de $2^{10}=1024$; fiecare intrare în această tabelă conține un pointer către o tabelă a nivelului 2(corespunzătoare lui Pt_2). Când o adresă virtuală este prezentată MMU, se extrage valoarea câmpului PT_1, care este folosit ca index în tabela de la nivelul 1. Pe baza acestuia, se găseşte adresa de început a uneia dintre tabelele de la nivelul *2*. Câmpul PT_2 va fi un index în tabela de la nivelul 2 selectată, de unde se va lua numărul de pagină fizică corespunzător numărului de adresă virtuală conținut în adresa referențiată. Pe baza acestui număr şi a deplasamentului d, se determină locația de memorie fizică în care va fi încărcată respectiva locație din memoria virtuală.

Observații.

1. Dimensiunea unei pagini este de $2^{12}=4x2^{10}=4K$.
2. În condițiile specificate, o tabelă de la nivelul 2 va gestiona o zonă de memorie de capacitate 1024 x 4 K=4 M..
3. Tabela de pagini se poate organiza şi pe 3 sau 4 niveluri, în funcție de dimensiunea memoriei virtuale şi a celei fizice. Dacă tabela are o dimensiune mare, ea poate fi păstrată pe un disc rapid.
4. Fiecare pagina virtuală conține un bit numit generic **present/absent,** pe baza valorii căruia se determină dacă pagina virtuală este (sau nu) adusă în memoria fizică.

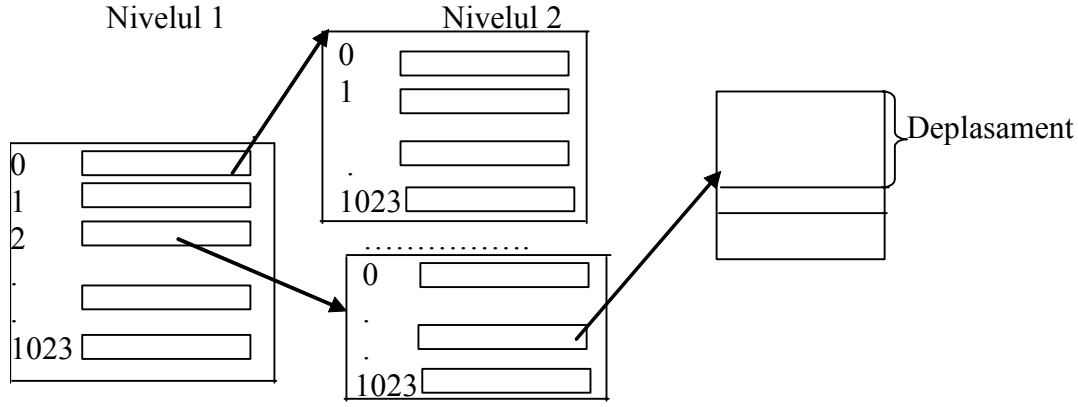


Figura 3.2.3. Tabela de pagini pe două niveluri.

Exemplu: Fie adresa virtuală 0x00A0B00C. Dacă se face transformarea în binar se obține: 0000000010 1000001011 0000 0000 1100.
Apoi se separă cele trei câmpuri şi se obțin valorile în zecimal.
$PT_1=0000000010_{(2)}=2_{(10)}$; $PT_2=1000001011_{(2)}=523_{(10)}$;
$d=000000001100_{(2)}=10$
Pe baza valorii lui PT_1, MMU va selecta a patra linie din tabela ne nivel 1, care conține un pointer către o tabelă de nivel 2 ; componenta PT_2 a adresei indică a 524-a intrare din tabela de la nivelul 2 selectată. Aceasta conține un pointer către o pagină din memoria fizică. Valoarea deplasamentului calculată, indică a 11-a locație din pagina respectivă, care corespunde adresei virtuale de adresă dată.

Să ne reamintim...

Paginarea memoriei a apărut la diverse SC pentru a evita fragmentarea memoriei interne, care apare la metodele anterioare de alocare. Memoria virtuală este împărțită în zone de lungime fixă numite **pagini virtuale.** Paginile virtuale se păstrează în memoria secundară. Memoria operativă este împărțită în zone de lungime fixă, numite **pagini fizice.** Paginile virtuale şi cele fizice au aceeaşi lungime, lungime care este o putere a lui *2* şi care este o constantă a sistemului.

Orice adresă paginată este formată din două componente: numărul de pagină şi deplasamentul în cadrul paginii.

Translatarea adreselor virtuale în adrese fizice se realizează pe baza unei structuri arborescente.

1. Presupunem că într-un sistem de calcul dimensiunea paginii este de *4 Ko*. Care este numărul de pagină şi deplasamentul pentru adresa 23955. Dacă locția de adresă dată este din memoria virtuală, descrieți translatarea ei într-o adresă fizică.
2. Presupunem că într-un sistem de calcul dimensiunea paginii este de 1 Ko. Care este numărul de pagină şi deplasamentul pentru următoarele adrese: 1099; 23955; 0x00FC1.

M3.U2.5. Alocare segmentată

Din punctual de vedere al utilizatorului**,** o aplicație este formată dintr-un program principal şi o mulțime de subprograme(funcții sau proceduri). Acestea folosesc diferite structuri de date (tablouri, stive etc), precum şi o mulțime de simboluri (variabile locale sau globale). Toate sunt identificate printr-un nume.

Pornind de la această divizare logică a unui program, s-a ajuns la o metoda alocării **segmentate** a memoriei. Spre deosebire de metodele de alocare a memoriei bazate pe partiționare, unde fiecărui proces trebuie să i se asigure un spațiu contiguu de memorie, mecanismul de alocare segmentată, permite ca un proces să fie plasat în zone de program distincte, fiecare dintre ele conținând o entitate de program, numit **segment**. Segmentele pot fi definite explicit prin directive ale limbajului de programare sau implicit prin semantica programului. Deosebire esențială dintre alocarea paginată şi cea segmentată este aceea că segmentele sunt de **lungimi diferite.**

Exemplu: Un compilator poate crea segmente **de cod** pentru fiecare procedură sau funcție, segmente pentru **variabilele globale,** segmente pentru **variabilele locale,** precum şi **segmente de stivă**.

În mod analog cu alocarea paginată, **o adresă virtuală** este o pereche `(s,d)`, unde `s` este numărul segmentului iar `d` este deplasamentul din cadrul segmentului. Adresa reală (fizică) este o adresă obisnuită. Transformarea unei adrese virtuale într-o adresă fizică, se face pe baza unei **tabele de segmente**. Fiecare intrare în acestă tabelă este compusă dintr-o **adresă de bază** (adresa fizică unde este localizat segmentul în memorie) şi **lungimea segmentului(limită)**. În figura următoare este ilustrat modul de utilizare a tabelei de segmente.

Componenta `s` a adresei virtuale, este un indice în tabela de segmente. Valoarea deplasamentului `d` trebuie să fie cuprinsă între `0` şi lungimea segmentului respectiv (în caz contrar este lansată o instrucțiune `trap`, care generează o întrerupere). Dacă valoarea `d` este validă, ea este adăugată adresei de început a segmentului şi astfel se obține adresa fizică a locației respective (figura 3.2.4).

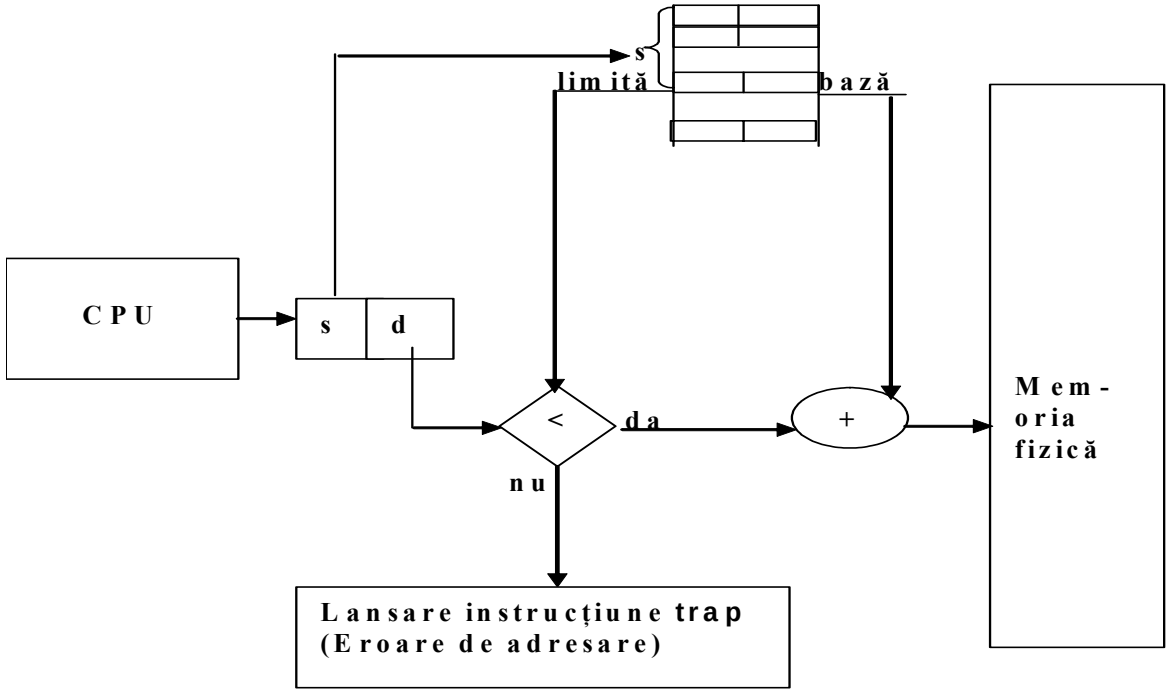
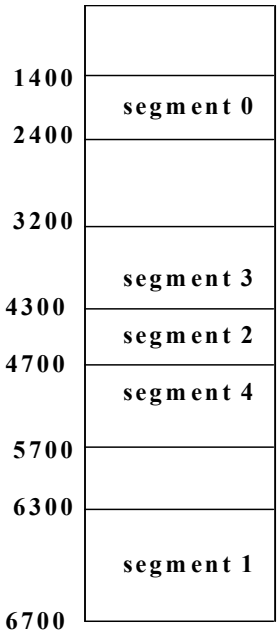


Figura 3.2.4. Translatarea unei adrese segmentate

Exemplu: Să presupunem că avem un proces care, din punct de vedere logic este format dintr-un program principal, un subprogram şi în care se apelează o procedură de sistem. Corespunzător celor trei entități vom avea trei segmente, la care se adaugă un segment pentru stivă şi unul pentru tabele de simboluri. Figura 3.2.5 redă alocarea de spațiu în memoria internă pentru acest proces. Segmentul `2` are o lugime de `400` octeți şi începe la locația `4300`. Astfel, unei referințe la locația `53` a segmentului `2`, îi corespunde locația din memoria internă `4500+53=4353`.

Număr segment	Limită	Bază
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

Tabela de segmente



Memoria fizică

Figura 3.2.5. Încărcarea segmentelor în memoria fizică

Avantajele alocării segmentate, față de cea bazată pe partiții sunt:

- se pot creea segmente reentrante, care pot fi partajate de mai multe procese. Pentru aceasta este suficient ca toate procesele să aibă în tabelele lor aceeaşi adresă pentru segmentul respectiv.

- se poate realiza o bună protecție a memoriei. Fiecare segment în parte poate primi alte drepturi de acces, drepturi trecute în tabela de segmente. De exemplu, un segment care conține instrucțiuni cod maşină, poate fi declarat "read-only" sau executabil. La orice calcul de adresă, se verifică respectarea modului de acces al locațiilor din segmental respectiv.

Alocarea segmentată cu paginare. Cele două metode de alocare a memoriei au avantajele şi dezavantajele lor. În cazul segmentării poate să apară fenomenul de fragmentare. În cazul paginării, se efectuează o serie de operații suplimentare de adresare. De asemenea, unele procesoare folosesc alocarea paginată (Motorola 6800), iar altele folosesc segmentarea (Intel 80x86, Pentium). Ideea segmentării cu paginare, este aceea că alocarea spațiului pentru fiecare segment să se facă paginat.

Spațiul de adrese virtuale al fiecărui proces este împărțit în două zone; prima zonă este utilizată numai de către procesul respectiv, iar cealaltă este partajată împreună cu alte procese. Informațiile despre prima partiție, respective a doua partiție sunt păstrate în descriptorul tabelei locale (LDT – Local Descriptor Table), respectiv descriptorul tabelei globale (GDT – Global Descriptor Table). Fiecare proces are propria LDT (caracter local), care conține informații despre segmentele locale ale procesului (cod, date, stivă etc); GDT este gestionată la nivel de sistem (caracter global) şi conține informații despre segmente de sistem, inclusiv ale sistemului de operare. Fiecare intrare în LDT/GDT conține informații despre un anumit segment, printre care adresa de început şi lungimea acelui segment.

Adresa virtuală este o pereche *(g,s,p,d)*, în care:
- *g* indică dacă segmentul este local sau global (se selectează LDT sau GDT);
- *s* este numărul de segment;
- *p* specifică pagina;
- *d* valoarea deplasamentului

La nivelul fiecărui segment, există o tabelă de translatare a adreselor virtuale în adrese fizice, organizată arborescent, pe trei niveluri, descrisă în secțiunea anterioară. Translatarea unei adrese virtuale în adresă fizică se realizează astfel:

- pe baza valorii *g* se alege tabela de segmente;
- pe baza valorii *s* se alege intrarea corespunzătoare segmentului, de unde se iau adresa de începuta şi lungimea segmentului;
- prin utilizarea tabelei de pagini, pe baza valorilor *p,d* se obține adresa fizică a locației virtuale respective.

Să ne reamintim...

Un segment reprezintă o entitate de program. Segmentele pot fi definite explicit prin directive ale limbajului de programare sau implicit prin semantica programului. Deosebire esențială dintre alocarea paginată şi cea segmentată este aceea că segmentele sunt de lungimi diferite**.**

În mod analog cu alocarea paginată, o adresă virtuală este o pereche *(s,d)*, unde *s* este numărul segmentului iar *d* este deplasamentul din cadrul segmentului. Adresa reală (fizică) este o adresă obisnuită. Transformarea unei adrese virtuale într-o adresă fizică, se face pe baza unei tabele de segmente.

Ideea segmentării cu paginare, este aceea că alocarea spațiului pentru fiecare

segment să se facă paginat. Această metodă este utilizată de sistemele de operare actuale.

Înlocuiți zona punctată cu termenii corespunzători.

1. Transformarea unei adrese virtuale într-o adresă fizică, se face pe baza unei ..... Fiecare intrare în acestă .....este compusă dintr-o ..... şi .....
2. Componenta .....a adresei segmentate virtuale, este un ..... în tabela de segmente.
3. Valoarea ..... d trebuie să fie cuprinsă între .....şi .....respectiv.
4. Într-o adresă virtuală segmentată cu paginare, g indică dacă segmentul este..... sau .....; s este .....; p specifică .....; d este .....

M3.U2.6. Algoritmi statici de paginare

Algoritmii statici alocă un număr fix de pagini fizice fiecărui proces în timpul execuției. Oricare algoritm de paginare este definit trei politici:

- **politica de extragere** (fetch): decide când o pagină va fi încărcată în memoria primară;
- **politica de înlocuire:** determină care pagină va fi mutată din memoria internă atunci când toate paginile acesteia sunt pline cu informații ale proceselor în execuție;
- **politica de plasare:** determină unde va fi plasată o pagină extrasă din memoria internă.

Deoarece numărul paginilor fizice este fixat în cadrul alocării statice, pagina fizică unde va fi adusă o pagină virtuală, va trebui să fie ori o pagină liberă, ori una în care se va înlocui informația deja existentă, care fac parte din spațiul de adrese al procesului respectiv. Deci politica de plasare este aceeaşi, celelate două politici diferențiază algoritmii statici de paginare.

Model matematic utilizat în descrierea algoritmilor statici de paginare. Presupunem că N este mulțimea paginilor din spațiul de adrese virtuale; ω este şirul referințelor de pagini, fiind o secvență de pagini din N, de forma:

$$\omega = r_1, r_2, \ldots, r_i, \ldots$$

care sunt referențiate de către proces în timpul execuției sale. Timpul virtual al procesului este indicele din şirul de referințe de pagini. Fie m numărul de pagini din spațiul de adrese fizice şi $S_i(m)$ mulțimea paginilor fizice încărcate. La momentul 0 avem $S_0(m) = \Phi$ şi la momentul virtual t, $S_t(m) = S_{t-1}(m) \bigcup X_t - Y_t$, X_t, respectiv Y_t fiind mulțimea paginilor extrase, respectiv înlocuite la timpul virtual t.

Politici de extragere. Aducerea unei pagini la cerere, este politica de extragere utilizată; deoarece nu se cunoaşte apriori evoluția unui proces, nu se poate stabili dinainte o politică de pre-extragere a paginilor. Există unele sisteme care utilizează metode de încărcare prin care se aduc **pagini în avans**. Astfel, odată cu o pagină se aduc şi câteva pagini vecine, în ipoteza că ele vor fi invocate în viitorul apropiat. **Principiul vecinătății** afirmă că adresele de memorie solicitate de un program nu se distribuie uniform pe întreaga memorie folosită, ci se grupează în jurul unor centre. Apelurile din jurul acestor centre sunt mult mai frecvente decât apelurile de la un centru la altul. Acest principiu sugerează o politică simplă de încărcare în avans a unor pagini. Se stabileşte o aşa zisă memorie de lucru compusă din câteva pagini. Atunci când se cere aducerea unei pagini virtuale în memoria executabilă, sunt încărcate câteva pagini vecine acesteia. În conformitate cu principiul vecinătății, este foarte probabil ca următoarele

referiri să fie făcute în cadrul memoriei interne. O **evidență statistică** a utilizării paginilor, poate furniza, cu o anumită probabilitate care vor fi paginile cerute în viitor.

Algoritmi de înlocuire. Dacă considerăm $\omega = r_1, r_2, \ldots, r_i, \ldots$ un şir de referire a paginilor şi $\{S_t(m)\}$ secvența de stări ale memoriei interne atunci

$$S_t(m) = S_{t-1}(m) \bigcup X_t - Y_t$$

în care X_t şi Y_t au semnificația următoare:

- dacă la momentul t procesul cere o pagină r_t care nu se găseşte în memoria fizică şi y_t este pagina care va fi înlocuită atunci $X_t = \{r_t\}$, $Y_t = \{y_t\}$;
- dacă la momentul t procesul cere o pagină r_t care se găseşte în memoria fizică atunci $X_t = \Phi$, $Y_t = \Phi$.

Problema care se pune, este cum să fie selectată pagina y_t care va fi înlocuită în memoria fizică.

Înlocuirea aleatoare înseamnă că y_t este aleasă cu o probabilitate $1/m$ dintre oricare paginile fizice alocate procesului. În practică, s-a dovedit că această metodă nu este eficientă.

Algoritmul optimal al lui Belady. La la un moment *t*, pentru încărcarea unei pagini r_t, pagina fizică y_t care urmează să fie înlocuită, va fi determinată astfel: dintre paginile r_{t+1}, $r_{t+2}, \ldots,$ care fac parte din şirul de referințe, se va alege acea pagină r_k pentru care diferența $k-t+1$, $k \geq t+1$, considerând primele apariții ale paginilor, este maximă, sau $t=max\{k/r_k \in S_{t-1}(m)\}$ sau cu alte cuvinte se înlocuieşte pagina care **nu va fi folosită pentru cea mai lungă perioadă de timp**.

Observație. Această metodă nu poate fi aplicată direct, deoarece presupune că se cunoaşte apriori care pagini vor fi solicitate de către procesul respectiv, fapt imposibil, deoarece evoluția unui program este determinată de datele de intrare. Aceast algoritm poate fi o sursă de inspirație pentru alte metode.

Exemplu: Presupunem că numărul de pagini fizice alocate unui proces este 3. Este prezentat un şir de referințe de pagini (prima linie), precum şi paginile fizice în care ele se încarcă, folosind algoritmul lui Belady. a II-a linie coresp. primei pagini fizice alocate, a III-a linie corespunde celei de-a II-a pagini fizice alocate, a IV-a linie corespunde celei de-a III-a pagini fizice alocate. Am marcat cu * atunci când are loc fenomenul de lipsă de pagină.

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1	
7*	7	7	2*	2	2	2	2	2	2	2	2	2	2	2	2	2	7*	7	7	pagini fizice
	0*	0	0	0	0	0	4*	4	4	0*	0	0	0	0	0	0	0	0	0	
		1*	1	1	3*	3	3	3	3	3	3	3	1*	1	1	1	1	1	1	

Primele 3 pagini din şirul de referință sunt încărcate fără probleme în memoria fizică. Când se ajunge la a patra pagină virtuală din şir (pagina 2), va fi înlocuită pagina 7, deoarece, în viitor, ea va fi ultima la care se va face o referință, dintre paginile care sunt încărcate în memoria fizică. A cincea pagină virtuală din şir este deja încărcată în memoria internă, deci nu va mai fi necesară înlocuirea unei pagini ş.a.m.d.

Înlocuirea în ordinea încărcării paginilor(FIFO). Se crează şi se întreține o listă a paginilor în ordinea încărcării lor. Un nod al listei conține numărul paginii virtuale încărcate, timpul virtual al încărcării şi un pointer către nodul următor (nul pentru ultimul nod). Această listă

se actualizează la fiecare **nouă încărcare de pagină** şi funcţionează pe principiul cozii. Atunci când se cere înlocuirea, este substituită prima (cea mai veche) pagină din listă care va fi şi ştearsă din listă.

Exemplu: În condiţiile aceluiaşi şir de referinţe şi aceluiaşi număr de pagini fizice alocate unui proces, ca în exemplul anterior este prezentat un şir de referinţe de pagini, precum şi paginile fizice în care ele se încarcă, folosind algoritmul FIFO.

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7*	7	7	2*	2	2	2	4*	4	4	0*	0	0	0	0	0	0	7*	7	7
	0*	0	0	0	3*	3	3	2*	2	3	3	3	2	2	2	2	2	0*	0
		1*	1	1	1	0*	0	0	3*	4	4	2*	1*	1	1	1	1	1	1

} pagini fizice

Prima încărcare de pagină virtuală, cu înlocuirea unei pagini fizice, are loc atunci când se încarcă prima dată pagina *2*; lista conţine paginile *7*,*0*, *1*, pagina înlocuită fiind *7* (prima din listă) şi lista va conţine paginile *0*,*1*,*2*. La următoarea înlocuire de pagină, când este adusă în memorie pagina virtuală 3, este înlocuită pagina *0* şi lista va conţine paginile *1*,*2*,*3* ş.a.m.d.

Înlocuirea paginii nesolicitate cel mai mult timp (LRU **L**east **R**ecently **U**sed) are la bază observaţia că o pagină care a fost solicitată mult în trecutul imediat, va fi solicitată mult şi în continuare şi invers. Problema este cum să se ţină evidenţa utilizărilor. Fiecare pagină are alocată o locaţie în tabela de pagini, care se incrementează la fiecare accesare a acesteia.. Atunci când se impune o înlocuire, este înlocuită pagina cu cea mai mică valoare a contorului. Dacă există mai multe pagini cu aceeaşi valoare minimă a numărului de accesări, se aplică metoda FIFO, în selectarea paginii care va fi înlocuită.

Exemplu: În condiţiile aceluiaşi şir de referinţe şi aceluiaşi număr de pagini fizice alocate unui proces, ca în exemplul anterior, este prezentat un şir de referinţe de pagini, precum şi paginile fizice în care ele se încarcă, folosind algoritmul LRU.

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7*	7	7	2*	2	2	2	4*	4	4	0*	0	0	1*	1	1	1	1	1	1
	0*	0	0	0	0	0	0	0	3*	3	3	3	3	3	0*	0	0	0	0
		1*	1	1	3*	3	3	2*	2	2	2	2	2	2	2	2	7*	7	7

}pagini fizice

Când este adusă pentru prima dată în memoria fizică pagina virtuală 2, este înlocuită pagina 7, deoarece ea nu a mai fost utilizată de cel mai mult timp; din acelaşi motiv, când este referenţiată pentru prima dată pagina 3, ea înlocuieşte pagina 1 ş.a.m.d.

Fiecare pagină fizică are asociaţi doi biţi; bitul **R** (referenţiere), primeşte valoarea *0* la încărcarea paginii. La fiecare referire a paginii, acest bit este pus pe *1*; periodic (constantă de sistem), bitul este pus iarăşi pe *0*. Bitul **M** (modificare), primeşte valoarea *0* la încărcarea paginii respective şi este setat cu *1* la scriere în pagină; bitul **M** indică dacă pagina trebuie sau nu salvată înaintea înlocuirii.

Metoda NRU(Not Recently Used). La fiecare valoare a timpului virtual, biţii **M** şi **R** împart paginile fizice în patru clase:

- clasa 0: pagini nereferite şi nemodificate;
- clasa 1: pagini nereferite (în intervalul fixat), dar modificate de la încărcarea lor;
- clasa 2: pagini referite dar nemodificate;
- clasa 3: pagini referite şi modificate.

Atunci când o pagină trebuie înlocuită, pagina respectivă se caută mai întâi în clasa 0, apoi în clasa 1, apoi în clasa 2 şi în sfârşit în clasa 3. Dacă pagina de înlocuit este în clasa 1 sau în clasa 3, conţinutul ei va fi salvat pe disc înaintea înlocuirii.

Observaţie. Algoritmul NRU se utilizează împreună cu metoda FIFO, în sensul că se aplică mai întâi NRU, iar în cadrul aceleiaşi clase se aplică FIFO.

Metoda celei de-a doua şanse este o extensie a metodei FIFO. Când se pune problema lipsei de pagină, se testează bitul R al primei pagini din coadă. Dacă acesta este 0, atunci pagina este înlocuită imediat. Dacă este 1, atunci este pus pe 0 şi pagina este pusă ultima în coadă, având ca valoare a timpului virtual, timpul când se cere încărcarea de pagină. Apoi căutarea se reia cu nouă pagină care a devenit prima din coadă.

Alocarea cadrelor. În cazul sistemelor care utilizează multiprogramarea, fiecărui proces trebuie să i se asigure un număr de pagini fizice, care nu pot fi utilizate de celelalte procese. Prin a**locare egală** se alocă tuturor proceselor acelaşi număr de pagini fizice. Prin **alocarea proporţională** se alocă proceselor un număr de pagini fizice direct proporţional cu dimensiunea spaţiului de memorie virtuală utilizat de procesul respectiv. O altă metodă de alocare ţine cont de **prioritatea proceselor**, în sensul că numărul paginilor fizice alocate unui proces, să fie direct proporţională cu prioritatea procesului. Cele două criterii de alocare proporţională se pot combina.

Performanţele sistemului de paginare. Atunci când un proces referenţiază o pagină virtuală care nu este prezentă în memoria fizică, sistemul execută o serie întreagă de operaţii, care au fost amintite în secţiunea 3. este evident că timpul de execuţie al procesului într-un sistem cu paginare, va fi mai mare decât timpul său de execuţie, în cazul în care va avea la dispoziţie un spaţiu de memorie suficient, astfel încât să nu se confrunte cu situaţia de „pagină lipsă” (pagină referenţiată care nu este încărcată în memoria fizică, şi care trebuie adusă în locul unei pagini încărcate în memoria fizică). Dacă apare frecvent situaţia că, la un anumit moment al execuţiei, o anumită pagină este evacuată pe disc, după care, la un interval mic de timp ea este din nou referenţiată, eficienţa utilizării CPU va scade.

Să ne reamintim...

Algoritmii statici alocă un număr fix de pagini fizice fiecărui proces în timpul execuţiei. Oricare algoritm de paginare este definit trei politici: politica de extragere, politica de înlocuire, politica de plasare.

Algoritmul optimal al lui Belady înlocuieşte pagina care nu va fi folosită pentru cea mai lungă perioadă de timp.

Înlocuirea în ordinea încărcării paginilor crează şi se întreţine o listă a paginilor în ordinea încărcării lor.

Înlocuirea paginii nesolicitate cel mai mult timp are la bază observaţia că o pagină care a fost solicitată mult în trecutul imediat, va fi solicitată mult şi în continuare şi invers.

Metoda NRU foloseşte biţii M şi R împart paginile fizice în patru clase. Atunci când o pagină trebuie înlocuită, pagina respectivă se caută mai întâi în clasa 0,

apoi în clasa 1, apoi în clasa 2 şi în sfârşit în clasa 3. Dacă pagina de înlocuit este în clasa 1 sau în clasa 3, conținutul ei va fi salvat pe disc înaintea înlocuirii.

Considerăm şirul următor de referințe al paginilor:

5,0,1,2,3,4,6,2,3,1,0,1,6,0,2,3,1,4,6.

Presupunem că se alocă procesului 2,3, respectiv 4 pagini fizice.

i) Să se descrie procesul de alocare a paginilor, folosind înlocuirea optimă.

ii) Aceeaşi problemă, dacă politica de înlocuire este FIFO.

iii) Aceeaşi problemă, dacă politica de înlocuire este LRU.

M3.U2.7. Algoritmi dinamici de paginare

Prin utilizarea algoritmilor statici de alocare, se poate ajunge la situația în care anumitor procese nu li se asigură suficientă memorie fizică, iar alte procese nu folosesc eficient memoria fizică alocată. Problema care se pune, este găsirea unei strategii prin care se aduc în memoria fizică numai acele pagini de care procesul va avea nevoie într-un viitor apropiat. Numărul de pagini fizice de care va avea nevoie un proces pentru execuția lui, nu va mai fi o constantă, fiind modificat în funcție de necesități.

Metoda setului de lucru(WS-Working **S**et) rezolvă această problemă. Ea are la bază modelul localității. **Localitatea** este un **set de pagini** care sunt utilizate împreună într-un mod activ. Un program este format din mai multe localități, iar execuția programului presupune trecere dintr-o localitate în alta.

Exemplu: Partea de instrucțiuni cod maşină care corespund unui subprogam; aceste instrucțiunui se află în pagini vecine, care după execuția lor (terminarea execuției subrutinei) pot fi evacuate din memorie şi posibilitatea referirii lor într-un viitor apropiat este improbabilă.

Când un proces trece dintr-o localitate în alta, se modifică atât paginile cerute de către proces, cât şi numărul de pagini fizice necesar procesului. Astfel, se poate considera că numărul de pagini fizice alocat procesului este o **entitate dinamică** ce se modifică în timpul execuției procesului. De asemenea, prin această metodă problema alocării de memorie fizică proceselor are caracter global.

Modelul matematic. Presupunem că există n procese care partajează memoria fizică a sistemului. Fie $m_i(t)$ cantitatea de memorie alocată procesului i la momentul virtual t. Avem $m_i(0)=0$ şi $\sum_{i=0}^{n-1} m_i(t) = k$, k fiind dimensiunea memoriei interne. Printr-un raționament similar cu cel de la metodele statice de paginare, înlocuind pe m cu $m_i(t)$ obținem:

$$S_0(m_0(t))=0$$

$$S_t(m_i(t)) = S_{t-1}(m_i(t)) \cup X_t - Y_t$$

$S_t(m_i(t))$ reprezintă starea memoriei la momentul t pentru procesul p_i ($S_t(m_i(0))=\Phi$). Dacă la momentul t procesul cere pagina r_t şi aceasta este deja încărcată în memoria fizică, atunci $X_t=Y_t=\Phi$; dacă pagina r_t nu este încărcară în memorie, atunci $Y_t = \{y_t\}$, sau pagina y_t va fi evacuată din memorie, dacă numărul de referințe de la referirea paginii y_t, până la cererea paginii curente r_t este mai mare sau egală cu o valoare constantă, notată cu Δ, ce reprezintă dimensiunea unei ferestere logice. $m_i(t)$ va reprezenta numărul de pagini fizice alocat procesului p_i şi va fi ajustat astfel:

- $X_t \neq \Phi$ şi $Y_t = \Phi$, atunci $m_i(t) = m_i(t-1) + 1$ (se alocă o pagină fizică);
- $X_t = \Phi$ şi $Y_t = \Phi$, atunci $m_i(t) = m_i(t-1)$;
- $X_t = \Phi$ şi $Y_t \neq \Phi$ atunci $m_i(t) = m_i(t-1) - 1$ (una dintre paginile fizice este luată procesului).

$S(m_i(t))$ rezultat este numit set de lucru al procesului p_i la momentul t.

Vom nota cu $W(t,\Delta)$ setul de lucru relativ la parametrul Δ şi la timpul virtual t al procesului. $W(t,\Delta)$ este mulţimea de pagini pe care procesul le-a cerut în ultimele Δ unităţi de timp virtual. Variabila Δ este un interval de timp virtual (fereastră) în care procesul este observat. Setul de lucru are proprietatea

$$W(t,\Delta+1) \supseteq W(t,\Delta)$$

adică dimensiunea setului de lucru este o funcţie ne-descrescătoare de Δ.

Exemplu: Considerăm Δ=7 şi şirul de referinţe prezentat în figura 3.2.6; la valorile timpului virtual al procesului t_1 =6 şi t_2=15, $WS(t_1)=\{0, 1, 2, 3\}$, $WS(t_2)=\{4, 5, 6, 7, 8\}$.

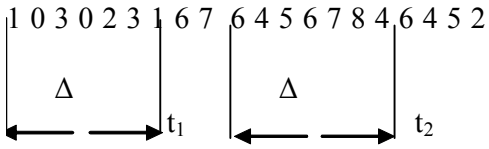
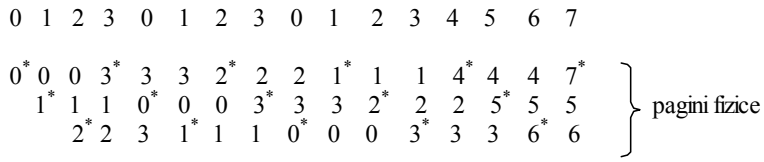
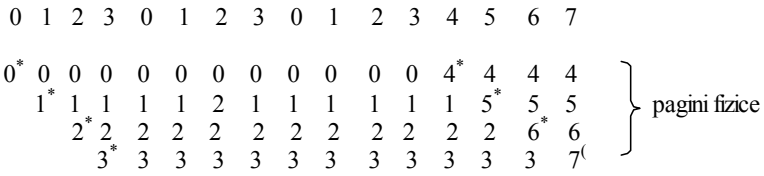


Figura 3.2.6. Valoarea setului de lucru în funcţie de fereastră

Exemplu: În figura 3.2.7 sunt prezentate diverse moduri de alocare a spaţiului de memorie fizică al procesului pentru diverse valori ale lui Δ.



a) Setul de lucru cu Δ=3.



b) Setul de lucru cu Δ=4.

Figura 3.2.7. Alocarea de pagini fizice prin metoda setului de lucru

Observăm că în cazul a) numărul de situaţii de „pagină lipsă” este 16, pe când în cazul b) acesta se reduce la 6.

Observaţie.

1. Cele două seturi de lucru prezentate în exemplul anterior sunt disjuncte, adică ele corespund la localităţi diferite.
2. Sistemul de operare alocă fiecărui proces un număr de pagini fizice corespunzător dimensiunii setului de lucru asociat; dacă suma dimensiunilor seturilor de lucru

corespunzătoare proceselor, este mai mică decât dimensiunea memoriei interne atunci sistemul mai poate executa încă un proces. Reciproc, dacă suma dimensiunilor seturilor de lucru ale proceselor depăşeşte dimensiunea memoriei interne, atunci unul dintre procese va fi suspendat.
3. În cursul execuției lor, procesele trec dintr-o localitate în alta. Atunci când se află într-o localitate, numărul paginilor utilizate (setul de lucru) tinde să se stabilizeze; atunci când procesele se află într-o stare de tranziție dintr-o localitate în alta dimensiunea setului de lucru creşte, datorită faptului că se fac noi referințe către pagini care corespund noii localități. Deoarece dimensiunea setului de lucru are o limită, se pune problema înlocuirii unor pagini din setul de lucru curent cu alte pagini.

Conceptul de set de lucru poate fi folosit pentru a realiza o strategie pentru modificarea mulțimii paginilor rezidente din memoria internă alocate procesului. Astfel:
- se monitorizează setul de lucru al fiecărui proces,
- periodic, se scot dintre paginile rezidente ale proceselor, cele care nu mai sunt în setul de lucru, pe baza unei strategii de tip LRU;
- un proces poate fi executat numai dacă setul său de lucru este în memoria internă, adică mulțimea paginilor rezidente din memoria internă include setul de lucru.

Deoarece ține cont de principiul localității, această strategie poate fi aplicată, conducând la minimizarea numărului situațiilor în care apare fenomenul de lipsă de pagină. Trebuie rezolvate anumite probleme legate de:
- dimensiunea setului de lucru şi paginile care fac parte din acesta se schimbă în timp;
- nu se poate estima setul de lucru al fiecărui proces în timp;
- nu se poate stabili o valoare optimă a parametrului Δ.

Următorul algoritm de actualizare a setului de lucru nu se bazează direct pe referințele de pagină, ci pe dimensiunea numărului de apariții(rata) a lipsei de pagină al procesului. Algoritmul **PFF** (**P**age **F**ault **F**requency) foloseşte un bit pentru fiecare pagină din memorie; acest bit este setat când pagina este accesată. Când apare o lipsă de pagină, sistemul de operare observă timpul virtual de la ultima lipsă de pagină a acelui proces. Pentru contorizarea acestuia, se foloseşte o variabilă care contorizează referințele la pagini. De asemenea, se foloseşte un prag notat cu F. Dacă valoarea timpului de la ultima lipsă de pagină este mai mic sau egal cu F, atunci pagina este adaugată mulțimii de pagini rezidente ale procesului din memorie; în caz contrar, se renunță la toate paginile cu bitul de utilizare setat pe 0 şi se micşorează corespunzător mulțimea paginilor rezidente. În acelaşi timp, se resetează bitul de utilizare al paginilor rezidente rămase ale procesului.

Observații.
1.Metoda poate fi îmbunătățită prin utilizarea a două praguri: un prag superior, utilizat pentru a declanşa o creştere a dimensiunii mulțimii paginilor rezidente şi un prag inferior, care este utilizat pentru diminuarea numărului paginilor rezidente.
2. Metoda prezentată are dezavantajul următor: deoarece paginile nu sunt eliminate din memorie decât după F unități de timp virtual, atunci când se trece dintr-o localitate în alta are loc o succesiune de lipsă de pagină, corespunzătoare noii localități, fără ca paginile corespunzătoare localității anterioare să fie eliminate; astfel mulțimea paginilor rezidente din memorie creşte şi unele dintre pagini nu mai sunt necesare.

Politica VSWS (**V**ariable-interval **S**ampled **W**orking **S**et) evaluează setul de lucru la anumite instanțe pe baza valoarii timpului virtual trecut. La începutul intervalului selectat, biții de utilizare ai tuturor paginilor rezidente ale proceselor sunt resetate; la sfârşit, numai paginile care au fost cerute pe parcursul intervalului vor avea bitul de utilizare setat; aceste pagini sunt reținute în setul rezident al procesului pe parcursul următorului interval, în timp ce la altele se

renunță. Astfel, la sfârșitul intervalului setul rezident poate numai să scadă. În decursul fiecărui interval, orice pagină lipsă este adăugată setului rezident; astfel, setul rezident poate să rîmână neschimbat sau să crească pe parcursul intervalului.

Strategia VSWS utilizează trei parametri:
M::durata minimă a intervalului de selecție;
L: durata maximă a intervalului de selecție;
Q: numărul situațiilor de lipsă de pagini care este permis să apară între două intervale de selecție.
Algoritmul VSWS este:
1. Dacă timpul virtual de la ultima instanță de selecție atinge L, atunci se suspendă procesul şi se scanează biții de utilizare.
2. Dacă, înainte de un timp scurs L, apar Q lipsă de pagină,
a**.** Dacă timpul virtual trecut de la ultima instanță de selecție este mai mic decât M, atunci aşteaptă până când timpul virtual trecut ajunge la valoarea M pentrua a suspenda procesul şi a scana biții de utilizare.
b. Dacă timpul virtual trecut de la ultima instanță de selecție este mai mare sau egal cu M, se suspendă procesul şi se scanează biții de utilizare.
Valorile parametrilor sunt luate astfel încât setarea intervalul de observație să fie declanşată după apariția a Q lipsă de pagină după ultima scanare.

Să ne reamintim...

Metoda setului de lucru rezolvă problema aducerii în memorie a paginilor de care are nevoie un proces la un moment dat. Ea are la bază modelul localității. **Localitatea** este un **set de pagini** care sunt utilizate împreună într-un mod activ. Un program este format din mai multe localități, iar execuția programului presupune trecere dintr-o localitate în alta.

Când un proces trece dintr-o localitate în alta, se modifică atât paginile cerute de către proces, cât şi numărul de pagini fizice necesar procesului. Astfel, se poate considera că numărul de pagini fizice alocat procesului este o **entitate dinamică** ce se modifică în timpul execuției procesului.

Vom nota cu $W(t,\Delta)$ setul de lucru relativ la parametrul Δ şi la timpul virtual t al procesului. $W(t,\Delta)$ este mulțimea de pagini pe care procesul le-a cerut în ultimele Δ unități de timp virtual. Variabila Δ este un interval de timp virtual (fereastră) în care procesul este observat**.**

I. Înlocuiți zona punctată cu termenii corespunzători.
1. Localitatea este un ..... care sunt utilizate împreună într-un mod activ. Un ..... este format din mai multe localități, iar execuția programului presupune ..... dintr-o localitate în alta.
2. Când un .....trece dintr-o localitate în alta, se modifică atât .....cerute de către ....., cât şi numărul de ..... necesar procesului.
3. Dacă $W(t,\Delta)$ este setul de lucru relativ la ..... Δ şi la ..... t al procesului, atunci $W(t,\Delta)$ este ..... pe care procesul le-a cerut în ultimele Δ .....

II. Considerăm şirul următor de referințe al paginilor:

1, 0, 1, 2, 3, 4, 5, 6, 7, 5, 6, 8, 1, 6, 5, 2, 3, 1, 0.

Să se descrie procesul de alocare a paginilor, folosind metoda setului de lucru, pentru $\Delta \in \{5,7\}$, dacă se alocă procesului 3, respectiv 4 pagini fizice.

M3.U2.9. Memoria cu acces rapid

Memoria cu acces rapid(„cache") conține copii ale unor blocuri din memoria operativă. Când CPU încearcă citirea unui cuvânt din memorie, se verifică dacă acesta există în memoria **cache**. Dacă există, atunci el este livrat CPU. Dacă nu, atunci el este căutat în memoria operativă, este adus în memoria cache împreună cu blocul din care face parte, după care este livrat CPU. Datorită vitezei mult mai mari de acces la memoria cache, randamentul general al sistemului creşte.

Memoria cache este împărțită în mai multe părți egale, numite **sloturi**. Un slot are dimensiunea unui bloc de memorie, a cărui dimensiune este o putere a lui *2*. Într-un slot se încarcă conținutul unui anumit bloc din de memorie operativă. Problema care se pune, este modul în care se face corespondența dintre blocurile din memoria operativă şi sloturile din memoria cache, precum şi politicile de înlocuire a sloturilor din memoria cache cu blocuri din memoria fizică. Spunem că are loc o **proiecție** a spațiului memoriei operative în cel al memoriei cache.

Proiecția directă. Dacă *c* indică numărul total de sloturi din memoria cache, *a* este o adresă oarecare din memoria operativă, atunci numărul *s* al slotului în care se proiectează adresa *a* este:

$$s = a \bmod c.$$

Dezavantajul metodei constă în faptul că fiecare bloc are o poziție fixă în memoria cache.

Exemplu: Dacă se cer accese succesive la două blocuri care ocupă acelaşi slot, atunci trebuie efectuate mai multe operații de înlocuire a conținutului slotului care corespunde celor două blocuri.

Proiecția asociativă. Fiecare bloc de memorie este plasat în oricare dintre sloturile de memorie cache, care sunt libere. Înlocuirea conținutului unui slot, cu conținutul altui bloc din memoria operativă, se face pe baza unuia dintre algoritmii NRU, FIFO sau LRU.

Proiecția set-asociativă combină cele două metode prezentate anterior. Memoria cache este împărțită în *i* seturi, un set fiind compus din *j* sloturi. Avem relația `c=i x j`. Dacă *a* este o adresă de memorie, numărul *k* al setului în care va intra blocul, este dat de:

$$K = a \bmod i.$$

Cunoscând numărul setului, blocul va ocupa unul dintre sloturile acestui set, pe baza unuia dintre algoritmii de înlocuire prezentați anterior.

M3.U2.9. Test de evaluare a cunoştințelor

Selectați varianta corectă.

1.Dacă presupunem că `n` este dimensiunea magistralei de adrese, atunci numărul maxim de locații adresabile este:

a) $2x2^n$. ☐ b) $nx2^n$. ☐

c) 2^n. ☐ d) 2^{n+1}. ☐

2. Spațiul de adrese al unui proces conține adrese din:

a) Memoria externă. ☐ b) Memoria internă. ☐

c) Memoria "cache". ☐ d) Memoria primară şi pe hard-discuri. ☐

3. Funcția de translatare a adreselor virtuale este:

a) O corespondență constantă în timp a spațiului de adrese virtuale ale unui proces, în spațiul de adrese fizice. ☐

b) O corespondență variabilă în timp a spațiului de adrese virtuale ale unui proces, în spațiul de adrese fizice. ☐

c) O corespondență variabilă în timp a spațiului de adrese virtuale ale unui proces, în spațiul de de adrese de pe disc. ☐

d) O corespondență constantă în timp a spațiului de adrese virtuale ale unui proces, în spațiul de de adrese de pe disc. ☐

4. O pagină virtuală este:

a) O zonă contiguă din memoria primară. ☐

c) O zonă contiguă dintr-un fişier executabil. ☐

b) Un slot din memoria "cache".. ☐

d) Un registru al CPU. ☐

5. Paginile virtuale şi cele fizice:

a) Au aceeaşi lungime, lungime care este o putere a lui *2* şi care sunt setate de administratorul de sistem. ☐

c) Au lungimi diferite, lungimi care sunt o putere a lui *2* şi care sunt setate de administratorul de sistem. ☐

b) Au aceeaşi lungime, lungime care este o putere a lui *2* şi care este o constantă a sistemului. ☐

d) Au lungimi diferite, lungimi care sunt o putere a lui *2* şi care este o constantă a sistemului.. ☐

6. Dacă 2^k este dimensiunea unei pagini, atunci numărul de pagină în care se află o locație de memorie de adresă `i` este:

a) Restul împărțirii lui i la 2^k. ☐

b) i – 2^k. ☐

c) Câtul împărțirii lui i la 2^k. ☐

d) i +2^k. ☐

7. În cazul calculatoarelor pe 32 de biți, tabela de translatare a adreselor virtuale în adrese fizice este organizată:

a) Pe 3 niveluri. ☐

b) Pe 2 niveluri.. ☐

c) Pe 4 niveluri. . ☐

d) Pe 5 niveluri.. ☐

8. O intrare în tabela de segmente este compusă din:

a) Adresă de bază şi lungimea segmentului. ☐

b) Adresă tabelei de pagini a segmentului şi lungimea segmentului. ☐

c) Adresă de bază şi lungimea paginilor segmentului. ☐

d) Adresă de bază şi lungimea fişierului executabil. ☐

9. Comform algoritmului optimal al lui Belady, se înlocuieşte pagina care:

a) Nu va fi folosită pentru cea mai lungă perioadă de timp. ☐

b) Nu va fi folosită pentru cea mai scurtă perioadă de timp. ☐

c)Este prima încărcată în memorie. ☐

d) Este ultima încărcată în memorie. ☐

10. Înlocuirea paginii nesolicitate cel mai mult timp are la bază observația că

a) O pagină care a fost solicitată mult în trecutul imediat, va fi solicitată puțin în continuare.. ☐

b) O pagină care a fost solicitată mult în trecutul imediat, va fi solicitată mult şi în continuare. ☐

c) O pagină care a fost solicitată mult la început, va fi solicitată mult şi în continuare. ☐

d) O pagină care va fi solicitată mult în viitor, nu a fost solicitată mult în trecut. ☐

11. Setul de lucru relativ la parametrul Δ şi la timpul virtual t al procesului, notat cu $W(t,\Delta)$ este:

a) Mulțimea de pagini pe care procesul le-a cerut în ultimele Δ unități de timp virtual. . ☐

b) Mulțimea de pagini pe care procesul le va cere în următoarele Δ unități de timp virtual. ☐

c) Mulțimea de pagini pe care procesul le-a cerut în primele Δ unități de timp virtual. ☐

d) Mulțimea de pagini pe care procesul le va încărca în următoarele Δ unități de timp virtual. ☐

12. În memoria "cache" se încarcă:

a) O pagină virtuală. ☐

b) Conținutul regiştrilor CPU. ☐

c) Un bloc din memoria internă ☐

d) Conținutul unui fişier executabil. ☐

M3.U2.10. Rezumat

Calculatoarele moderne folosesc conceptul de memorie virtuală, ce reprezintă extinderea spațiului de adrese al memoriei fizice. Ea se sprijină pe ideea asocierii de adrese fizice obiectelor programului în timpul execuției lui. Compilatorul şi link-editorul creează un modul absolut, căruia încărcătorul îi asociază adrese fizice înainte ca programul să fie executat. Facilitățile hardware permit administratorului memoriei să încarce automat porțiuni ale spațiului de adrese virtuale în memoria primară, în timp ce restul spațiului de adrese este păstrat în memoria secundară.

Sistemele cu paginare transferă blocuri de informație de dimensiune fixă între memoria secundară şi cea primară. Datorită dimensiunii fixe a paginii virtuale şi a celei fizice, translatarea unei adrese virtuale într-una fizică devine o problemă relativ simplă, prin mecanismul tabelei de pagini sau al memoriei virtuale.

Politicile de încărcare şi de înlocuire a paginilor, permit ca paginile să fie încărcate în memorie, numai atunci când se face o referință la o locație de memorie pe care o conțin. Politicile de plasare şi de înlocuire a paginilor, definesc reguli prin care se optimizează transferul de informații din memoria virtuală în memoria primară şi reciproc.

Segmentarea este o alternativă la paginare. Ea diferă de paginare prin faptul că unitățile de transfer dintre memoria secundară şi cea primară variază în timp. Dimensiunea segmentelor trebuie să fie în mod explicit definită de către programator sau sistem. Translatarea unei adrese virtuale segmentate într-o adresă fizică este mult mai complexă decât translatarea unei adrese virtuale paginată. Segmentarea se sprijină şi pe sistemul de fişiere, deoarece segmentele sunt stocate pe memoriile externe sub formă de fişiere. Segmentarea cu paginare valorifică avantajele oferite de cele două metode, fiind utilizată de sistemele de calcul moderne.

Memoriile cu acces rapid cresc performanțele unui sistem de calcul. Metodele de proiecție a spațiului din memoria internă în memoria cache, permite ca orice locație fizică să fie accesată mult mai rapid, pentru a fi utilizată de unitatea centrală.

Modulul 4. Administrarea fişierelor

Cuprins

Unitatea de învăţare M4.U1. Administrarea fişierelor

Cuprins

Durata medie de parcurgere a unităţii de învăţare este de 3 ore.

Introducere.

Multe aplicaţii care se execută pe un sistem de calcul prelucrează un volum mare de informaţii. În timpul execuţiei sale, un proces poate reţine o parte din informaţiile pe care le prelucrează în spaţiul său de adrese virtuale. Acest spaţiu este limitat ca dimensiune şi conţinutul lui se şterge atunci când se încheie o sesiune de lucru. O altă problemă este necesitatea partajării de către procese a unor volume mari de informaţii.

Sistemul de operare, prin componenta sa de gestiune a fişierelor rezolvă aceste probleme, prin stocarea datelor pe suporturi externe de informaţie sub formă de fişiere. Această structură de date permite efectuarea de diverse operaţii (adăugare, modificare, ştergere, localizare etc.), care vor fi prezentate în continuare.

De asemenea, sistemul de fişiere oferă o serie de apeluri de sistem prin care se oferă o serie de facilităţi de lucru cu fişiere, care ascund detaliile de implementare, uşurând astfel munca utilizatorului.

Unele sisteme de operare privesc dispozitivele de I/O ca fişiere, în scopul de a simplifica utilizarea acestora.

Obiectivele unităţii de învăţare

La sfârşitul acestei unităţii de învăţare studenţii vor fi capabili:

- să explice conceptul de fişier;
- să explice operaţiile care se pot efectua asupra fişierelor;
- să explice modul de organizare a fişierelor;

- să identifice diferența dintre discul fizic şi cel logic;
- să explice conceptul de director;
- să explice metodele de gestionare a spațiului liber pe discuri;
- să identifice interacțiunea dintre sistemul de fişiere şi celelalte componente ale sistemului de operare.

M4.U1.3. Conceptul de fişier

Din punctul de vedere al sistemului există două modalități de a gestiona spațiile de memorie secundară: utilizarea **memoriei virtuale** şi **sistemul de fişiere**. Dacă memoria virtuală este utilizată de către un proces în timpul execuției, fişierele sunt folosite pentru stocarea informațiilor pentru o perioadă mai lungă de timp. Intuitiv, un fişier este un ansamblu de elemente de date, grupate împreună, având ca scop controlul accesului, regăsirea şi modificarea elementelor componente. Aceste elemente de date pot fi **octeți** sau **articole**(înregistrări). În figura 4.1.1. se prezintă transformările pe care le suferă datele stocate pe memorii secundare pentru a putea fi utilizate de către procese.

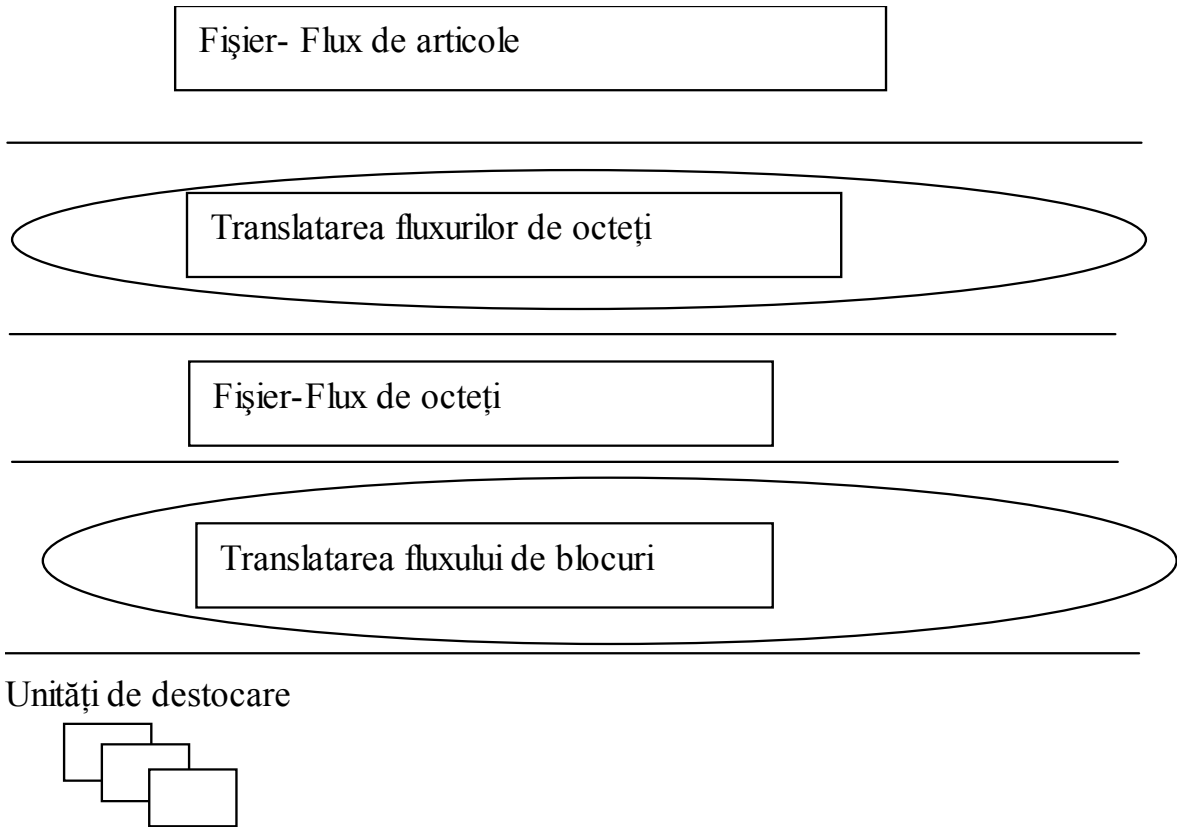


Figura 4.1.1. Nivele de structurare a informației

La nivelul unui periferic, datele sunt stocate ca secvențe de blocuri de informație. Pentru a fi utilizate de către aplicații, informațiile sunt structurate sub formă de fişiere; componenta SO care se ocupă de gestionarea lor se numeşte sistemul de fişiere. Când sunt folosite de aplicații, se presupune că fişierele sunt structurate sub forma unei colecții de articole şi la rândul lor, informațiile dintr-un articol sunt de regulă grupate în subdiviziuni, numite **câmpuri** sau **atribute**. Unele sisteme de operare furnizează numai facilități de translatare a blocurilor în fluxuri de octeți, lăsând în seama aplicațiilor structurarea fişierelor ca flux de articole. Astfel de sisteme de operare conțin un **sistem de fişiere de nivel scăzut**. Sistemele de operare

Windows, Linux şi Unix fac parte din această clasă. Sistemele de operare care privesc fişierele ca flux de înregistrări conţin un **sistem de fişiere de nivel înalt**.

Fiecare câmp(atribut) al unui articol, are o anumită **valoare** în cadrul fişierului. O pereche **(atribut, valoare)** o vom numi cheie. Numim **index de articol,** un atribut cu proprietatea că pentru oricare două articole diferite ale fişierului, valorile atributului sunt diferite. Indexul de articol se mai numeşte şi **cheie de articol**.

Exemplu. Considerăm un fişier care conţine informaţii despre studenţii unei universităţi (numele, prenumele, numărul matricol, data naşterii, locul naşterii, seria si numărul buletinului de identitate, sexul, facultatea din care face parte, notele obţinute la examene etc.). Perechile (nume, ”Ionescu”), (sex, B), (matricol, `1089`) sunt chei. Se observă că în acest fişier pot exista mai multe articole care conţin aceeaşi cheie (de exemplu mai mulţi studenţi cu acelaşi nume) şi de asemenea, pot fi chei care să nu existe în fişier. Atributele **număr matricol** precum şi atributul obţinut prin concatenarea câmpurilor **seria si numărul buletinului de identitate** reprezintă indexi**.**

De cele mai multe ori, numărul de câmpuri dintr-un articol este acelaşi pentru toate articolele fişierului. Numărul de octeţi pe care se reprezintă un atribut al lui, este constant sau variabil.

Exemplu. Un atribut care este un numar întreg se reprezintă pe doi octeţi, indiferent de articol, dar un atribut care este un şir de caractere poate avea lungimi diferite de reprezentare.

Lungimea de reprezentare a unui articol poate fi constantă pentru toate articolele fişierului sau poate să varieze de la un articol la altul, deci avem articole de **format fix** şi articole de **format variabil**. Pentru cele cu format variabil, trebuie să existe fie un câmp în care se trece lungimea efectivă a articolului, fie o valoare specială prin care să se marcheze terminarea articolului.
Conceptul de fişier abstract. Un fişier `f` poate fi privit ca o funcţie parţial definită $f:N \to T$, unde `N` este mulţimea numerelor naturale, iar `T` este mulţimea valorilor posibile pentru un anumit tip de dată deja definit.

Exemplu. T poate fi mulţimea valorilor posibile pentru date reprezentate pe un octet, şi atunci avem un fişier flux de octeţi, poate fi un tip de date şi atunci avem un fişier structurat ca un flux de articole; dacă tipul de dată este la rândul lui un tip fişier, avem de-a face cu o **bază de date.**

Un întreg `i` ($i \in N$) indică numărul de ordine al unui articol(octet) din fişier; primul articol are numărul de ordine `0`, al doilea are numărul de ordine `1` ş.a.m.d.; prin `f(i)` notăm valoarea articolului al `i`-lea, adică ansamblul valorilor câmpurilor acestui articol.

Descriptorul de fişier este o structură de date gestionată de SGF în care sunt păstrate informaţii despre un anumit fişier. Această structură diferă de la un sistem de operare la altul, dar sunt unele informaţii comune conţinute şi anume:
• **Numele extern(simbolic)** este un şir de caractere utilizat pentru a identifica un fişier în comenzile utilizatorilor sau în programe. Orice fişier se identifică prin perechea `(N,I)`, unde `N` reprezintă numele simbolic al fişierului iar `I` este un număr prin care descriptorul este reperat pe disc în mod direct. Prin `N`, sistemul de fişiere realizează legătura cu utilizatorul, iar prin `I` cu celelalte componente ale SO.
• **Starea curentă** poate fi:

- **arhivat**, dacă el nu poate fi deschis fără a se efectua un număr de operaţii asupra sa;

- **închis,** dacă el se află pe un suport magnetic on-line, şi poate fi deschis întrun timp suficient de scurt;
- **deschis** pentru citire(scriere, execuţie etc.) dacă ele este alocat unui proces care poate efectua una dintre operaţiile specificate.

• **Partajare**. Un câmp care specifică faptul că mai multe procese pot utiliza în acelaşi timp fişierul respectiv, putând efectua una dintre operaţiile specificate mai sus.
• **Proprietar**. Conţine identificatorul asociat utilizatorului al cărui proces a creat fişierul. Sistemul permite ca dreptul de proprietate să fie transferat şi altor utilizatori.
• **Utilizator.** Conţine lista proceslor care la momentul respectiv au deschis fişierul.
• **Încuiat.** SO poate să „încuie ” fişierul atunci când acesta este deschis.

Exemplu. Dacă fişierul este partajat, numai procesul respectiv poate utiliza fişierul atâta timp cât acesta este încuiat.

• **Protecţie.** Conţine nişte fanioane care indică modul în care fişierul respectiv poate fi folosit de către diverse clase de utilizatori.
• **Lungime.** Numărul de octeţi(articole) conţinute în fişierul respectiv.
• **Data creării (data ultimei modificări, data ultimei accesări)**. Conţine datele sistem ale creării, ultimei modificări, respectiv ultimei accesări a fişierului repectiv.
• **Contorul de referenţiere.** Dacă fişierul este partajat el conţine numărul de procese care au deschis fişierul respectiv simultan.
• **Detalii privind modul de stocare.** Indică modul cum blocurile fişierului pot fi accesate, în funcţie de modul de administrare a unităţii respective.

Clasificarea fişierelor se poate face după mai multe criterii:
I. **După lungimea unui articol,** se disting:
• fişiere cu articole de **format fix;**
• fişiere cu articole de **format variabil.**
II. **După posibilitatea de afişare sau tipărire a conţinutului** se disting: fişiere **text** şi fişiere **binare**.
Fişierele text sunt cele al căror conţinut poate fi afişat pe ecran sau poate fi tipărit la imprimantă. El este format dintr-o succesiune de octeţi, fiecare conţinând codul unui caracter tipăribil. Codificarea caracterelor se face utilizând unul dintre sistemele ASCII sau EBCDIC.
Articolul unui fisier text este **caracterul**. Într-un fişier text, o linie este un şir de caractere, care se termină cu un caracter special numit **separator de linii**, format de regulă din două caractere funcţionale: CR si LF.
Fişierele binare sunt formate din şiruri de octeţi consecutivi fără nici o semnificaţie pentru afişare.

Exemplu. Fişierele obiect rezultate în urma compilării şi fişierele executabile rezultate în urma editării de legături sunt fişiere binare.

III. După **suportul pe care este rezident** fişierul(fişiere pe hard disc; fişiere pe floppy disc; fişiere pe imprimantă; fişiere de la tastatură; fişiere pe monitor.)
Evident, că nu toate aceste tipuri de fişiere acceptă orice operaţii şi moduri de acces.

Exemplu. Numai suportul disc acceptă accesul direct, celelalte numai cel secvenţial. Fişierele de la tastatură acceptă numai citirea iar fişierele pe imprimantă numai scrierea.

IV. După modurile de acces permise de către un fişier se disting :
• **fişiere secvenţiale** care permit numai accesul secvenţial la înregistrări;

• **fişiere cu acces direct** permit accesul direct la înregistrări, cel puțin la citire.

Să ne reamintim...
Intuitiv, un fişier este un ansamblu de elemente de date, grupate împreună, având ca scop controlul accesului, regăsirea şi modificarea elementelor componente. Aceste elemente de date pot fi **octeți** sau **articole**(înregistrări). Informațiile dintr-un articol sunt de regulă grupate în subdiviziuni, numite **câmpuri** sau **atribute**. Numim **index de articol,** un atribut cu proprietatea că pentru oricare două articole diferite ale fişierului, valorile atributului sunt diferite. **Descriptorul de fişier** este o structură de date gestionată de SGF în care sunt păstrate informații despre un anumit fişier.

Înlocuiți zona punctată cu termenii corespunzători.

1. La nivelul unui periferic, datele sunt stocate ca secvențe de .....
2. Când sunt folosite de aplicații, se presupune că fişierele sunt structurate sub forma unei .....
3. Numim **index de articol,** un atribut cu proprietatea că pentru oricare două articole ..... ale fişierului, valorile atributului sunt .....
4. Un fişier `f` poate fi privit ca ....., unde `N` este ....., iar `T` este mulțimea valorilor posibile pentru .....
5. Prin `f(i)` notăm valoarea .....
6. **Descriptorul de fişier** este o structură de date gestionată de ..... în care sunt păstrate .....
7. **Articolul** unui fisier text este **.....**. Într-un fişier text, o linie este un şir de caractere, care se termină cu .....

M4.U1.4. Operații asupra fişierelor

Fişierul este un tip de date abstract, deci asupra lui se pot efectua o serie de operații; sistemul de operare oferă o serie de apeluri de sistem, prin care se pot lansa aceste operații şi al căror efect va fi descris în cele ce urmează.

Deschiderea fişierului poate fi realizată **explicit**, printr-un apel de sistem (**open)** sau **implicit** la prima referire a fişierului**.** Sistemul de operare păstrează o **tabelă** a fişierelor deschise şi fiecărui fişier deschis îi corespunde un indice în această tabelă. Când este lansată o operație asupra fişierului, acesta este selectat pe baza valorii indicelui din tabelă, evitându-se astfel efectuarea unor operații de căutare mult mai complexe. Operația **open** returnează un pointer către intrarea corespunzătoare din tabela fişierelor deschise. Acest pointer va fi utilizat în toate operațiile de I/O care se vor efectua asupra fişierului.
Efectul operației open este diferit în funcție de natura fişierului. Există acțiuni comune, indiferent de tipul fişierului. **Atât pentru un fişier nou creat, cât şi unul existent**:

- se face legătura dintre identificatorul logic, utilizat de program şi descriptorul de fişier aflat pe disc;
- se face alocarea de memorie internă pentru zonele tampon necesare accesului la fişier;
- se încarcă rutinele de acces la articolele.
- sunt efectuate o serie de controale asupra drepturilor de acces ale utilizatorului la fişier.

În plus**, pentru un fişier nou creat** se alocă spațiu pe disc pentru memorarea viitoarelor articole ale fişierului.

Închiderea fişierului se poate realiza **implicit,** la terminarea execuției procesului sau **explicit** prin lansare unui apel de sistem **(close)**. Şi aici se poate discuta efectul operației de închidere,

în funcție de tipul fişierului. Pentru **fişierele temporare** se şterge fişierul şi se eliberează spațiul pe disc ocupat. P**entru toate fişierele care trebuie reținute,** se efectuează următoarele acțiuni:
- se actualizează informațiile generale despre fişier (lungime, adrese de început şi de sfârşit, data modificării etc.);
- se aduce capul de citire al discului în poziția zero;
- se eliberează spațiile ocupate de zonele tampon în memoria operativă;
- se eliberează perifericul sau perifericele suport ale fişierului.

În plus**, pentru fişierele nou create şi care trebuie reținute**, se efectuează operațiile:
- se creează o nouă intrare în directorul discului (concept pe care îl vom aborda mai târziu);
- se inserează marcajul de sfârşit de fişier (EOF) după ultimul articol al fişierului;
- se goleşte tamponul adică ultimele informații existente în zonele tampon, sunt transferate pe periferic;

Dacă fişierul a fost modificat, se marchează acest lucru şi eventual se pune numele fişierului într-o listă a sistemului, în vederea unei salvări automate de către SO a tuturor fişierelor modificate.

Observație. Într-un mediu **multiusers** mai mulți utilizatori pot deschide acelaşi fişier în acelaşi timp şi, deci implementarea operațiilor **open** şi **close** devine mult mai complicată. Pentru rezolvarea acestei probleme, sistemul de operare foloseşte o tabelă pe **două niveluri**: o **tabelă la nivel de proces** şi una la **nivel de sistem**.

Tabela asociată procesului ține evidența tuturor fişierelor deschise de procesul respectiv şi aici sunt stocate informații care sunt utilizate numai de proces în lucrul cu fişierul respectiv(drepturile de acces la fişier, valoarea pointerului care indică următorul articol ce poate fi citit/scris etc.). În tabela de sistem, sunt memorate informații care privesc partajarea fişierului respectiv de către mai multe procese. Sistemul administrează un **cont de deschidere**, care este inițializat cu `1`, când un proces deschide fişierul, este incrementat de fiecare dată când un alt proces deschide acelaşi fişier, respectiv decrementat, atunci când un proces execută un apel **close** asupra fişierului respectiv. Atunci când contorul devine `0`, înseamnă că fişierul nu mai este folosit de nici un proces, deci intrarea corespunzătoare lui va fi ştearsă din tabelă.

C**itirea(Read)** celui de-al `k`-lea articol din fişier înseamnă obținerea valorii `f(k)`; `k` trebuie să fie valoarea indicatorului de fişier.

Scrierea(Write) înseamnă adăugarea unui articol la sfârşitul fişierului. Deci fişierul `f` cu `n` articole se transformă într-un fişier `f1` cu `n+1` articole, definit astfel:

$$f1(i) = f(i), \text{daca } 0 \le i \le n$$

$$f1(n+1) = x$$

unde `x` este valoarea articolului care trebuie introdus. Evident, valoarea `x` trebuie anterior să fie inițializată.

Blocarea şi deblocarea articolelor. Aşa cum am menționat, pentru prelucrarea informațiilor dintr-un fişier, trebuie efectuată o operație de citire, prin care acestea sunt aduse în memoria internă. De asemenea, pentru salvarea unor informații într-un fişier trebuie efectuată o operație de scriere. Cele două operații sunt costisitoare din punctul de vedere al sistemului. Deci, este indicat ca printr-o operație de citire/scriere să fie aduse în memoria internă, respective transferate pe disc, nu un articol ci mai multe articole. La nivel de disc, fişierul este divizat în unități(blocuri) fizice, care au aceeaşi dimensiune. Un astfel de bloc fizic, poate conține un număr de articole ale fişierului. Un fişier se poate diviza astfel în **blocuri** logice**.** Blocul este unitatea de schimb între suportul fişierului şi memoria internă. Zona de memorie

în care este adus un bloc pentru a fi prelucrat, respectiv unde sunt păstrate informațiile care urmează să fie transferate în fişiere se numeşte **zonă tampon**(buffer).

Pentru fişierele pe disc, se cere ca blocurile să fie compuse dintr-un număr întreg de sectoare vecine. Mai mult, aceste sectoare trebuie să fie în întregime cuprinse într-o pistă sau într-un cilindru. Fiecare SO are modalități proprii de stabilire a dimensiunii blocului.

Observații.
1. Citirile/scrierile economisite sunt înlocuite cu mutări ale unor octeți dintr-o zonă de memorie în alta.
2. Prin blocarea articolelor poate apare fenomenul de **fragmentare internă** al discului, datorită cerinței ca într-un bloc să intre un număr întreg de articole.
3.În cazul unui fişier în care se scriu articole noi, este obligatorie închiderea acestuia, operație care va transfera şi informațiile transferate în zona tampon, după ultima operație de scriere.

Inserarea(Insert) a unui nou articol cu valoarea `x`, după articolul cu numărul de ordine `k` înseamnă obținerea unui nou fişier `f1` cu `n+1` articole, definit astfel:

$$f1(i) = f(i), daca\ \ 0 \le i \le k$$
$$f1(k+1) = x$$
$$f1(i+1) = f(i), daca\ \ k \le i \le n$$

Se observă că operația de scriere este echivalentă cu operația de inserare dupa ultima poziție, notată cu `n`. Prin definiție, inserarea după poziția `0`, înseamnă plasarea unui articol nou la începutul fişierului. **Inserarea** unui articol trebuie făcută în aşa fel încât celelalte articole să rămână nemişcate. Acest fapt este posibil numai dacă articolele sunt legate între ele printr-o listă înlănțuită. Aceasta permite ca două articole să fie vecine logic, chiar dacă ele ocupă spații fizice îndepărtate.

Ştergerea(Delete) articolului `k` înseamnă obținerea unui nou fişier `f1` cu `n-1` articole, definit astfel :

$$f1(i) = f(i), daca\ 1 \le i \le k-1\}$$
$$f1(i-1) = f(i), daca\ \ k+1 \le i \le n\}$$

Ştergerea poate fi **logică** sau **fizică**. Prin ştergerea logică articolul respectiv continuă să ocupe zona fizică. Pentru aceasta, sistemul de fişiere adaugă la fiecare articol câte un octet numit **indicator de ştergere.** Dacă valoarea lui este 1, înseamnă că articolul există pentru sistemul de fişiere, în caz contrar el este ignorat de sistem, considerându-l şters. Pentru ştergerea fizică, trebuie recreat fişierul.
Observații.
1. Operațiile efectuate asupra fişierului, nu construiesc de fiecare dată câte un nou fişier. Sistemul de fişiere actionează local asupra informațiilor de pe suport şi lasă nemişcate articolele neimplicate în operație.
2. Operațiile descrise anterior se execută prin lansarea unor apeluri de sistem. Operațiile care vor fi prezentate în continuare, se execută ă prin intermediul unor comenzi (fişiere de comenzi) ale SO.

Crearea unui fişier. Pentru fişierele cu organizare mai simplă, această operație coincide cu deschiderea unui fişier nou în vederea scrierii în el. Pentru fişierele cu organizare mai complicată crearea înseamnă o formatare, care pregăteşte spațiul fizic în vederea încărcării lui cu informații, cum este cazul bazelor de date.

Ştergerea unui fişier. Atunci când informațiile dintr-un fişier nu mai sunt necesare, utilizatorul poate cere o astfel de operație. Ca efect, se eliberează spațiul fizic ocupat de

articolele fişierului şi se elimină din director intrarea care conţine descriptorul fişierului respectiv.

Copierea unui fişier în alt fişier prin care se realizează o copie fizică a fişierului **emiţător** sau **sursă** într-un fişier **receptor** sau **destinaţie**. Cele două fişiere pot să se găsească pe acelaşi suport sau pe suporturi diferite. Elementele de identificare ale celor două fişiere sunt diferite, deşi conţinuturile lor sunt aceleaşi. **Listarea unui fişier** este un caz particular de copiere, atunci când fişierul sursă este un fişier care poate fi tipărit, iar fişierul destinaţie este imprimanta sau ecranul unui terminal.

Schimbarea numelui unui fişier (redenumirea sau mutarea) lasă neschimbat conţinutul fişierului, dar îi schimbă elementele de identificare.

Prin **concatenarea** mai multor fişiere $f_1, \ldots, f_n$(fişierele sursă) se obţine un nou fişier f, în care conţinutul fişierului f_{i+1} urmează pe cel al lui f_i .

Compararea se face parcurgând secvenţial cele două fişiere, octet cu octet. După depistarea unui număr fixat de nepotriviri (de multe ori 10) compararea se opreşte cu rezultat fals. De regulă, dacă cele două fişiere nu au aceiaşi lungime sau sunt de acelaşi tip, comparaţia nu se lansează.

Sortarea (ordonarea) articolelor unui fişier, se realizează după valorile unuia sau mai multor câmpuri. Este utilizată foarte frecvent, mai ales în aplicaţiile cu caracter economic.

Să ne reamintim...
Deschiderea fişierului poate fi realizată **explicit**, printr-un apel de sistem (**open)** sau **implicit** la prima referire a fişierului**. Închiderea fişierului** se poate realiza **implicit,** la terminarea execuţiei procesului sau **explicit** prin lansare unui apel de sistem **(close)**. C**itirea(Read)** celui de-al `k`-lea articol din fişier înseamnă obţinerea valorii `f(k`. **Scrierea(Write)** înseamnă adăugarea unui articol la fişier.

La nivel de disc, fişierul este divizat în unităţi(blocuri) fizice, care au aceeaşi dimensiune. Un astfel de bloc fizic, poate conţine un număr de articole ale fişierului. Un fişier se poate diviza astfel în **blocuri** logice**.** Blocul este unitatea de schimb între suportul fişierului şi memoria internă.

Înlocuiţi zona punctată cu termenii corespunzători.

1. Deschiderea fişierului poate fi realizată **.....**, printr-un ..... sau implicit la ..... a fişierului**.**
2. Operaţia **open** returnează ..... către intrarea corespunzătoare din tabela fişierelor deschise.
3. Închiderea fişierului se poate realiza ……., la terminarea execuţiei …….. sau ……. prin lansare unui apel de sistem (close).
4. Citirea(Read) celui de-al `k`-lea articol din fişier înseamnă obţinerea valorii .....
5. Scrierea(Write) înseamnă adăugarea unui articol la .....
6. Un fişier se poate diviza astfel în .....logice**.** Blocul este ..... între suportul fişierului şi memoria internă.
7. Zona de memorie în care este adus un bloc pentru a fi prelucrat, respectiv unde sunt păstrate informaţiile care urmează să fie transferate în fişiere se numeşte .....

M4.U1.5. Moduri de organizare a fişierelor

Tipuri de acces la articole. Să presupunem că există un fişier `f` şi se efectuează numai operaţii de citire asupra lui. **Accesul secvenţial** la un articol `f(i)`, presupune realizarea anterior în ordine a `i-1` accese, la articolele cu numerele de ordine `0, 1,…,i-1`. **Accesul direct(random access)** presupune existenţa unui mecanism de obţinere a articolului căutat, fără parcurgerea prealabilă a tuturor articolelor care îl preced. Mecanismele de acces direct sunt:

• Acces direct prin **număr de poziţie (adresă),** are loc atunci când se furnizează sistemului de fişiere o valoare `i` şi aceasta întoarce o valoare `f(i)` a articolului reaspectiv.

• Acces direct prin **conţinut** are loc atunci când se furnizează SO o cheie `(a,v)`, adică o valoare `v` pentru un atribut `a` şi aceasta întoarce acel articol `i` pentru care are loc relaţia `f(i).a=v`, adică atributul `a` al articolului are valoarea căutată `v`.

Organizarea secvenţială presupune că între articolele fişierului este stabilită o relaţie de ordine astfel: un articol `X` ”urmează” altui articol `Y`, dacă `X` a fost introdus în fişier după introducerea lui `Y`. Sistemul de fişiere scuteşte utilizatorul de a vedea aspectul fizic al reprezentării. Formatul articolelor la un fişier secvenţial poate fi **fix** sau **variabil.** Accesul la un articol al unui fişier secvenţial se face, de regulă, tot secvenţial.

Fişiere cu acces direct prin poziţie. Un astfel de fişier are articole de format fix şi el este stocat pe hard-disc. Pentru fişierele cu acces direct, operaţiile de citire-scriere trebuie să conţină, ca parametru numărul blocului. Acest număr este furnizat de către utilizator sistemului de operare şi este un indice relativ la începutul fişierului, primul bloc relativ este `1`, al doilea este `2`, ş.a.m.d. Dacă lungimea unui articol este `l` şi se cere articolul cu numărul `n`, atunci printr-o operaţie de I/O vor fi accesaţi `l` octeţi, începând cu locaţia `lx(n-1)`. De asemenea, presupunem că articolele sunt plasate pe suport unul după celălalt, chiar dacă un articol începe pe un sector şi se termină pe altul.

Observaţii.

1. În cazul formatului variabil, această schemă nu mai funcţionează. Există însă diverse metode de acces direct şi în acest caz, bazate fie pe o listă a adreselor de pe disc unde începe fiecare articol, fie pe lista lungimilor acestor articole.

2. Majoritatea implementărilor de limbaje de programare de nivel înalt au mecanisme de acces direct prin poziţie pentru articole de format fix (funcţia **seek**).

Organizarea secvenţial indexată a fişierelor se bazează pe conceptul **index**; **indexul** este un **atribut** cu valori unice pentru fiecare articol. Deci, această metodă este caracterizată de accesul prin conţinut.

Articolele sunt scrise pe suport în acces secvenţial şi sunt grupate în blocuri logice de informaţie(**pagini),** astfel încât valorile indexului tuturor articolelor dintr-o pagină sunt mai mari sau mai mici decât valorile indexului articolelor dintr-o altă pagină. Odată cu crearea fişierului, se creează şi o **tabelă de indecşi**. Pentru fiecare pagină, în această tabelă, se memorează adresa de pe disc a paginii şi valoarea maximă a indecşilor din pagină.

Scopul acestui mod de organizare, este de **a înlocui căutarea secvenţială** a articolelor, cu **căutarea binară** sau **arborescentă**. Prin aceste metode de căutare se determină pagina în care se găseşte articolul respectiv, după care se efectuează o căutare secvenţială în pagina respectivă. Din aceste motive, în funcţie de dimensiunea fişierului, tabela de indecşi poate fi o listă liniară sau o structură arborescentă. Astfel, distingem următoarele cazuri:

i) Fişierul este memorat pe mai multe volume(partiţii) de disc; în acest caz, vom avea o tabelă organizată pe trei niveluri.

Nivelul 3(fişier), are atâtea intrări câte volume disc ocupă fişierul şi fiecare intrare are forma :

Ultimul index din volum	Numărul volumului

Tabela este plasată pe ultimul volum al fişierului.
Nivelul 2(volum), are atâtea intrări câți cilindri sunt alocați fişierului în volumul respectiv şi are forma :

Ultimul index din cilindru	Numărul cilindrului

Tabela este plasată pe ultimul cilindru alocat fişierului în volumul respectiv.
Nivelul 1 – cilindru, are atâtea intrări câte pagini din fişier există pe cilindrul respectiv, şi are forma :

Ultimul index din pagină	Adresa paginii

Tabela este plasată pe ultimele sectoare ale cilindrului.
ii) Fişierul este memorat pe un singur volum de disc; în acest caz, vom avea o tabelă organizată pe două niveluri, care au aceeaşi descriere cu nivelurile `1` şi `2` descrise anterior.
iii) Fişierul este memorat pe un singur cilindru; în acest caz, vom avea o tabelă ce reprezintă o listă.

Observaţie. Pentru căutarea unui articol, numărul de accesări ale discului depinde de dimensiunea fişierului şi de locul unde este memorată tabela respectivă.

Exemplu. Dacă fişierul este multivolum şi tabela de nivel 3 este încărcată în memoria internă, atunci pentru găsirea paginii unde se află articolul respectiv sunt necesare 5 accesări ale discului.

Pentru se putea realiza **actualizarea** unui fişier secvenţial – indexat, spaţiul fizic alocat unei pagini logice este divizat în două părți: **partea principală** şi **partea de depăşire.** Articolele din partea de depăşire sunt organizate ca o listă înlănțuită**.** Astfel, pentru **inserarea** unui articol, dacă corespunzător cheii respective, articolul nu mai are loc în partea principală, el va fi introdus în partea de depăşire. Dacă ar exista un spaţiu fizic de dimensiune fixă, corespunzător unei pagini logice, atunci ar exista posibilitatea ca articolul cu cheia respectivă să nu mai aibă loc în pagina fizică unde ar trebui introdus, deci ar trebui reorganizată tabela de indecşi. Dacă se fac mai multe înserări între două chei vecine din partea principală, atunci lista leagă mai multe articole în partea de depăşire, iar randamentul accesului la disc scade considerabil.

Ştergerile de articole contribuie şi ele la scăderea randamentului global al fişierului. Prin ştergere, fie că rămâne un loc nefolosit în partea principală, fie trebuie reorganizată o listă simplu înlănțuită în partea de depăşire.

Din motivele prezentate, rezultă necesitatea **reorganizării periodice** a unui astfel de fişier. Această reorganizare presupune:
- rescrierea întregului fişier;
- eliminarea porțiunilor şterse;
- articolele din partea de depăşire trec în partea principală;
- se reface tabela de indecşi.

Acest gen de fişiere prezintă două mari dezavantaje:
- n**ecesitatea reorganizării** lor destul de frecvente;
- necesitatea **efectuării de două până la cinci accese la disc** pentru un articol.

Organizarea selectivă a fişierelor. Se presupune că articolele fişierelui conţin un index. Fie `T` tipul de date al indexului. Articolele fişierului se împart în `n` **clase de sinonimie**. Clasa de sinonimie căreia îi aparţine articolului, se obţine pe baza unei funcţi parţial definite:

$$f : T \rightarrow \{0,1,\ldots, n-1\}$$

numită **funcţie de regăsire(randomizare).** Toate articolele pentru care se obţine aceeaşi valoare a funcţiei de randomizare, fac parte din aceeaşi clasă de **sinonimie.** Deci, organizarea selectivă foloseşte **accesul direct prin conţinut**. O **grupare logică** de articole(clasă de sinonimie) trebuie să corespundă unui bloc fizic de stocare pe disc.

Fişierele selective sunt o alternativă la fişierele indexat secvenţiale, în sensul următor: în timp ce pentru fişierele indexat–secvenţiale căutarea paginii în care se află articolul respectiv se face prin tabelele de index, în cazul celor selective, pagina este clasa de sinonimie, care se determină aplicând funcţia de regăsire, ce reprezintă un calcul efectuat de CPU, deci, mult mai rapid.

Dezavantajul acestei metode, constă în faptul că definirea funcţiei de regăsire este foarte puternic dependentă de datele ce vor fi înmagazinate în fişierul respectiv.

Actualizarea fişierelor selective se realizează într-un mod analog celui al fişierelor indexat-secvenţiale. **Scrierea** se realizează astfel: Se aplică funcţia de randomizare valorii indexului şi se obţine numărul clasei(de sinonime) în care se află articolul. **Partea principală** a fişierului este formată din câte o pagină pentru fiecare clasă. Dacă pagina clasei respective nu este încă plină, atunci articolul se depune în pagina respectivă pe disc. Dacă pagina din partea principală este plină, atunci articolul este pus în **partea de depăşire**. Ca şi la fişierele secvenţial–indexate, acest articol este legat printr-o listă simplu înlănţuită de ultimul articol sinonim cu el din partea principală.

La **citire**, utilizatorul formează indexul articolului dorit a fi citit. sistemul de fişiere îi aplică acestuia funcţia de randomizare, depistând clasa de sinonimie din care face parte. În cadrul clasei, căutarea articolului se face secvenţial, mai întâi în partea principală şi apoi în partea de depăşire.

Exemplu. Să presupunem că se doreşte crearea unui fişier selectiv pentru un colectiv de `2000` de persoane. Fiecare persoană are un cod unic de identificare, exprimat printr-un număr între `1` şi `2500`. Presupunem că se foloseşte drept suport un disc cu `10` piste pe cilindru, iar pe o pistă încap `20` de articole. Vom alege drept pagină fizică o pistă, deci aproximativ `20` de articole într-o clasă de sinonimie, numărul claselor de sinonimie `n=100`, iar funcţia de randomizare este:

$$f(x)=x \pmod{100}$$

Dacă, valorile indexului sunt `100, 200,.., 2000, 2100, 2200, 2300, 2400, 2500` (multiplu de `100`), înseamnă că toate aceste articole sunt sinonime (fac parte din clasa `0`) şi sunt în număr de `25`. Deci, această clasă va avea `5` articole plasate în partea de depăşire, număr care este destul de mare mare şi care poate creşte dacă sunt adăugate articole în fişier al căror indice să fie multiplu de `100`.

Exemplu. Presupunem că fişierul are `10000` de articole, însă indexul se reprezintă printr-un cod de `20` de cifre. S-ar impune o împărţire în `500` de clase, însă de această dată delimitarea superioară a părţii de depăşire pentru o clasă practic nu se poate face. Este posibil ca toate cele 10000 de articole să fie sinonime, deci căutarea s-ar face într-o singură clasă, adică am avea de fapt un fişier secvenţial.

Observaţie. Din cele două exemple prezentate, rezultă că eficienţa căutării unui articol într-un fişier selectiv este dependentă de datele pe care le conţine.

Să ne reamintim...
Accesul secvenţial la un articol presupune accesarea anterioară a articolelor cu numere de ordine `mai mici`. **Organizarea secvenţială** presupune că între articolele fişierului este stabilită o relaţie de ordine astfel: un articol `X` "urmează" altui articol `Y`, dacă `X` a fost introdus în fişier după introducerea lui `Y`. **Accesul direct(random access)** presupune existenţa unui mecanism de obţinere a articolului căutat, fără parcurgerea prealabilă a tuturor articolelor care îl preced.

În cazul organizarea secvenţial indexate a fişierelor, pe baza tabelei de indexi, se determină pagina în care se găseşte articolul respectiv, folosind **căutarea binară** sau **arborescentă; apoi,** se efectuează o căutare secvenţială în pagina respectivă pentru a se găsi înregistrarea de index dat.

Organizarea selectivă a fişierelor este o alternativă la fişierele indexat secvenţiale, în sensul următor: în timp ce pentru fişierele indexat–secvenţiale căutarea paginii în care se află articolul respectiv se face prin tabelele de index, în cazul celor selective, pagina este clasa de sinonimie, care se determină printr- un calcul efectuat de CPU, deci, mult mai rapid.

Înlocuiţi zona punctată cu termenii corespunzători.

1. Accesul secvenţial la un articol `f(i)`, presupune realizarea anterior în ordine a **.....** accese, la articolele cu numerele de ordine .....
2. Accesul direct presupune existenţa unui mecanism de obţinere a articolului căutat, fără ..... a tuturor articolelor care îl preced.
3. Organizarea secvenţială presupune că între articolele fişierului este stabilită o ..... astfel: un articol `X` "urmează" altui articol `Y`, dacă `X` a fost introdus în fişier ..... lui `Y`.
4. Organizarea secvenţial indexată a fişierelor se bazează pe conceptul .....
5. Scopul organizării secvenţial indexate, este de a înlocui ..... a articolelor, cu .....
6. Tabela de indexi este organizată pe trei niveluri, dacă fişierul este memorat pe…...
7. În cadrul organizării secvenţial indexate a fişierelor, spaţiul fizic alocat unei pagini logice este divizat în două părţi: ..... şi .....
8. În cadrul organizării ..... clasa de sinonimie căreia îi aparţine articolului, se obţine pe baza .....
9. Dezavantajul **organizării selective a fişierelor**, constă în faptul că definirea funcţiei de regăsire este ..... ce vor fi înmagazinate în fişierul respectiv.

M4.U1.6. Conceptul de director(catalog)

În secţiunea anterioară am abordat problema găsirii unui mecanism cât mai eficient de accesare a informaţiilor dintr-un fişier. În continuare, ne vom ocupa de problema regăsirii cât mai rapide a fişierelor de pe un disc.

Orice disc este împărțit în **partiții logice** sau **discuri virtuale** (una sau mai multe). Reciproc, există posibilitatea de a grupa mai multe discuri fizice întrun disc logic. Împărțirea discului în partiții, oferă utilizatorilor o serie de avantaje, cum ar fi:

- posibilitatea de funcționare a sistemului respectiv sub mai multe sisteme de operare;
- când se face reinstalarea sistemului de operare, nu se mai pierd prin formatarea discului toate informațiile salvate pe discul respectiv, ci numai cele din partiția respectivă;
- fiecare partiție conține o **tabelă** cu informații despre fișierele care le conține, accesului la aceste fișiere; astfel de tabelă poartă numele de **catalog** sau **director** și fiecare intrare înter-un director conține un **descriptor de fișier**.

În cele ce urmează, vom studia diferite **structuri de directoare**, care au apărut odată cu evoluția sistemelor de operare. Orice astfel de structură trebuie să respecte următoarele cerințe:

- c**ăutarea unui fișier.** O structură de directori conține câte o intrare pentru fiecare fișier, în care se află descriptorul de fișier (sau un pointer către el). Operația trebuie să permită identificarea unui fișier sau a unui grup de fișiere care fac parte din această familie;
- **inserare.** La crearea unui nou fișier, în sistemul de directori se va introduce o nouă intrare, care conține descriptorul fișierului respectiv;
- **ștergere.** La ștergerea unui fișier se șterg din directori informațiile relative la fișierul respectiv.
- **listare.** Fiecare SO dispune de comenzi care afișează conținutul directorilor.
- **salvarea-restaurarea** de directori, împreună cu fișierele subordonate(cele care corespund unor intrări în directorul respectiv);
- **parcurgerea** directorului, adică posibilitatea accesării fiecărui fișier din director.

Scheme de organizare a unui a unui sistem de directori. Sunt cunoscute patru tipuri de structuri: structuri liniare(un singur nivel); structuri pe două niveluri; structuri arborescente; structură de graf aciclic.

Structura de directoare pe un singur nivel. Avem un singur director la nivelul discului și fiecărui fișier îi este asociată o intrare în acest director. Această organizare are dezavantajul că limitează spațiul de nume al fișierelor, deoarece numele fișierelor trebuie să fie diferite, chiar dacă fișierele respective aparțin unor utilizatori diferiți. Deci, numărul de fișiere este limitat și în cazul redenumirii unui fișier, există pericolul ca noul nume să fie comun pentru două fișiere.

Structura de directoare pe două niveluri. La primul nivel se află directorul **Master** (MFD - **M**aster **F**ile **D**irectory). Acesta conține câte o intrare pentru fiecare utilizator al sistemului, intrare care punctează spre un **director utilizator**(UFD - User File Directory). Pentru fiecare nou utilizator al sistemului, se creează o intrare în MFD și când un utilizator se loghinează la sistem, se face o căutare în UFD, pentru a se găsi directorul corespunzător numelui său. De fiecare dată când un utilizator se referă la un fișier, căutarea se va face numai în directorul asociat lui. Toate aceste probleme sunt rezolvate de către sistemul de operare, prin intermediul sistemului de fișiere.

Această structură rezolvă problema limitării spațiului de nume al fișierului, dar prezintă și un mare dezavantaj; nu există posibilitatea comunicării între utilizatori, deci nu există posibilitatea partajării resurselor între utilizatori. O altă problemă apare atunci când un utilizator dorește să utilizeze o componentă software de sistem. Pentru utilizarea acesteia, trebuie realizată o copie, aflată în directorul utilizatorului respectiv.

Structura de directoare arborescentă. Structura de directori pe două niveluri, poate fi privită ca un arbore a cărui adâncime este 2. În mod natural, s-a pus problema generalizării acestei structuri, adică de a reprezenta structura de directori ca un arbore de adâncime

arbitrară. Această generalizare permite utilizatorilor să creeze propriile lor structuri de directori. Sistemele de operare moderne folosesc o astfel de structură. Aceste SO împart fişierele în: **obişnuite** şi **directori. Sistemul Unix** foloseşte şi un alt tip de fişiere, şi anume cel **special.**

Indiferent de sistemul de operare utilizat, **fişierele obişnuite** sunt privite ca şiruri de înregistrări sau succesiuni de octeţi, accesul putându-se realiza prin unul din mecanismele prezentate. Sub Unix, dispozitivele de I/O sunt privite ca fiind fişiere **speciale**. Din punctul de vedere al utilizatorului, nu există deosebiri între lucrul cu un fişier disc obişnuit şi lucrul cu un fişier. Această abordare are o serie de avantaje:
- simplitatea şi eleganţa comenzilor;
- numele unui dispozitiv poate fi transmis ca argument în locul unui nume de fişier;
- fişierele speciale sunt supuse aceluiaşi mecanism de protecţie ca şi celelalte fişiere.

Un **fişier director** se deosebeşte de un fişier obişnuit numai prin informaţia conţinută în el. Un director(sau subdirector) conţine lista de nume şi adrese pentru fişierele sau subdirectoare subordonate lui. De obicei, fiecare utilizator are un director propriu care punctează la fişierele lui obişnuite sau alţi subdirectori definiţi de el. Toate directoarele au acelaşi format intern. O intrare într-un director are forma:

Numele fişierului subordonat	Descriptorul fişierului(pointer spre el)

Pentru a se face distincţia dintre un subdirector şi un fişier obişnuit, se foloseşte un bit care este setat pe 1 pentru un fişier, respectiv 0 pentru un subdirector.

Fiecare director are două intrări cu nume special şi anume :
- “ . “ (punct) care punctează spre însuşi directorul respectiv;
- “ . . “ (două puncte succesive), care punctează spre directorul părinte.

Fiecare volum disc conţine un director iniţial numit **root**(rădăcină). Într-o sesiune, la orice moment, utilizatorul se află într-un **director curent,** care conţine majoritatea fişierelor utilizate la un moment dat. De asemenea, la intrarea în sistem, fiecare utilizator se află într-un **director gazdă (home directory**. SO permite utilizatorului să-şi schimbe directorul curent, să-şi creeze un nou director, să afişeze calea de acces de la **root** la un director sau fişier etc. Orice fişier este identificat prin **calea** către el care, care este un şir de directoare separate printr-un caracter, ce poate porni de la rădăcină(cale absolută) sau de la directorul curent(cale relativă).

Observaţii.
1. Directoarele cu structură de arbore au fost introduse pentru a înlocui căutarea secvenţială a fişierelor pe un disc cu căutarea arborescentă.
2. Pe lângă faptul că acest tip de directori nu mai limitează dimensiunea spaţiului de nume al fişierelor de pe un disc, permite utilizatorului să-şi structureze propria ierarhie de directoare în conformitate cu anumite criterii, să păstreze fişierele în anumite directoare, în funcţie de conţinutul lor.
3. Sistemele de operare oferă o serie întreagă de operaţii asupra directorilor: creare, mutare dintr-un director în alt director, copierea unei structuri în alta, ştergere etc.

Directori cu structură de graf. Această organizare este utilă, atunci când un fişier trebuie să fie accesibil din mai mulţi directori părinte, ceea ce permite partajarea unui fişier de către mai mulţi utilizatori. Această structură este specifică sistemului Unix.

Să ne reamintim...

Orice disc este împărțit în **partiții logice** sau **discuri virtuale**. Reciproc, există posibilitatea de a grupa mai multe discuri fizice întrun disc logic. Fiecare partiție conține o **tabelă** cu informații despre fișierele care le conține; o astfel de tabelă poartă numele de **catalog** sau **director** şi fiecare intrare înter-un director conține un **descriptor de fişier**.

Directorul pe un singur nivel este o tabelă şi fiecărui fişier îi este asociată o intrare în acest director.

În cazul d**irectorului pe două niveluri, l**a primul nivel se află directorul **master**; acesta conține câte o intrare pentru fiecare utilizator al sistemului, intrare care punctează spre un **director utilizator**(UFD - User File Directory). Pentru fiecare nou utilizator al sistemului, se creează o intrare în MFD şi când un utilizator se loghinează la sistem, se face o căutare în UFD, pentru a se găsi directorul corespunzător numelui său.
În cazul structurii arborescente, există un director rădăcină al discului respectiv; acesta are câte o intrare pentru fiecare fişier sau sub-director. La rândul lor, sub-directoarele conțin intrări pentru alte fişiere şi directoare ş.a.m.d. Directorii cu structură de arbore au fost introduşi pentru a înlocui căutarea secvențială a fişierelor pe un disc cu căutarea arborescentă.

Directorul cu structură de graf este utilă, atunci când un fişier trebuie să fie accesibil din mai mulți directori părinte, ceea ce permite partajarea unui fişier de către mai mulți utilizatori.

1. Orice disc este împărțit în ..... sau .....
2. Structura de directoare pe un singur nivel presupune că avem un ..... şi fiecărui fişier îi este asociată o ..... în .....
3. Structura de directoare pe două niveluri presupune că ..... se află ..... Acesta conține câte o intrare pentru fiecare ....., intrare care punctează spre un .....
4. Orice fişier este identificat prin ..... către el care, care este un şir de ..... separate printr-un caracter, ce poate porni de la ..... sau de la ..... curent.
5. Directoarele cu structură de arbore au fost introdue pentru a înlocui .....a fişierelor pe un disc cu .....

M4.U1.7. Alocarea spațiului pentru fişiere disc

Alocarea contiguă cere ca fiecare fişier să ocupe un set de blocuri consecutive de pe disc. Deoarece adresele de pe disc ale blocurilor care compun fişierul sunt în ordine, pentru a accesa blocul de adresă `b`, după blocul `b-1`, nu necesită mutarea capului de citire, cu excepția cazului când se trece de la ultimul sector al unui cilindru, la primul sector al următorului cilindru. Astfel, pentru accesarea directă a blocurilor fişierului, numărul de mutări ale capului discului este minimal.

Dacă presupunem că fişierul conține `n` blocuri şi adresa primului bloc este `b`, atunci fişierul va ocupa blocurile de adrese `b`, `b+1`, ...,`b+n-1`; deci, în descriptorul de fişier din director,

trebuie să se memoreze adresa de început şi lungimea zonei alocate. De asemenea, un astfel de fişier acceptă atât accesul secvenţial, cât şi cel aleator.

Problemele ridicate de acest fel de alocare, precum şi modalităţile lor de rezolvare, coincid cu cele care apar la alocarea dinamică a memoriei operative. Şi aici apare fenomenul de **fragmentare**. Opţional şi aici se efectuează **compactarea**, în aceleaşi condiţii şi cu aceleaşi metode ca şi cele întâlnite la gestiunea memoriei.

Alocarea înlănţuită rezolvă problemele legate de alocare contiguă (fenomenul de fragmentare). Fiecare fişier este privit ca o listă înlănţuită de blocuri de pe disc. Directorul conţine câte un pointer către primul, respectiv ultimul bloc al fişierului şi fiecare bloc conţine un pointer către următorul bloc al fişierului. Valoarea pointerului pentru ultimul bloc este `nil`. Toate operaţiile de găsire a blocurilor libere necesare fişierului, precum şi realizarea legăturilor între blocuri sunt realizate de către sistemul de operare.

Avantajul aceastei metode este evitarea fragmentării discului. Dezavantajele acestei metode sunt:
- nu permite decât accesul secvenţial la fişiere.
- se consumă **spaţiu de pe disc** pentru pointeri;
- **fiabilitatea -** pointerii se află pe tot discul şi ştergerea unui pointer poate duce la distrugerea fişierului; o soluţie pentru evitarea acestui dezavantaj, este gruparea blocurilor într-o nouă unitate de alocare numită **cluster**.

Exemplu. Sistemul de fişiere poate defini clusterul ca având 4 blocuri.

Această metodă are **avantajul** că diminuează spaţiul necesar memorării pointerilor, că micşorează numărul acceselor la disc atunci când se citesc/scrie informaţii de pe/pe disc. Totuşi, această metodă măreşte gradul de fragmentare al discului.

Exemplu. Ordinea blocurilor pe care este memorat fişierul este: 9, 12, 1, 2, 11, 22, 25. În director, în intrarea corespunzătoare fişierului, este trecut blocul 9 ca prim bloc al fişierului şi blocul 25, ca ultim bloc al fişierului. Fiecare bloc va conţine un pointer către următorul.

O variantă a acestei metode, presupune utilizarea unei tabele de alocare a fişierelor**(FAT-F**ile **A**location **T**able**),** aflată la începutul fiecărui disc logic. Tabela este indexată după numărul de bloc şi fiecare intrare va conţine numărul blocului care urmează în fişier, blocului al cărui număr este index, dacă acest bloc este ocupat. Indexului ultimului bloc din fişier, îi corespunde o valoare specială, de sfârşit de fişier(-1). Blocurile neutilizate sunt marcate în tabelă cu valoarea 0. Pentru alocarea unui nou bloc unui fişier, se caută prima intrare din tabelă care are valoarea 0, şi se înlocuieşte valoarea anterioară de sfârşit de fişier, cu adresa noului bloc, iar valoarea 0 este înlocuită cu marca de sfârşit de fişier.

Exemplu. Presupunem că avem un disc cu `12` blocuri, în care sunt memorate două fişiere(fis1, fis2). Presupunem că, iniţial fişierul fis1 ocupă blocurile `5, 2, 4` iar fis2 blocurile `6, 3, 9`. În figura 4.3.1.a sunt prezentate intrările în director, şi conţinutul **FAT** pentru această situaţie. Dacă fişierul fis1, va mai necesita un bloc, acesta va fi blocul 1, tabela **FAT** fiind cea din figura 4.3.2.b.

Director

fis1	...	5
fis2	...	6

	FAT		FAT
1	0	1	-1
2	4	2	4
3	9	3	9
4	–1	4	-1
5	2	5	2
6	3	6	3
7	0	7	0
8	0	8	0
9	–1	9	1
10	0	10	0
11	0	11	0
12	0	12	0
	a)		b)

Figura 4.3.1. Structura unei tabele FAT

Alocarea indexată. Fiecare fişier are o **tabelă de index,** în care se trec în ordine crescătoare adresele tuturor blocurilor ocupate de fişierul respectiv. Cea de-a `i`-a intrare din tabela de index conţine un pointer către cel de-al `i`-lea bloc al fişierului; această tabelă se păstrează într-un bloc separat. În intrarea corespunzătoare fişierului din director, se păstrează această tabelă. Această tabelă poate avea o organizare arborescentă, în funcţie de dimensiunea fişierului, similară tabelei utilizate în paginarea memoriei interne. Dacă fişierul este mare şi ocupă mai multe adrese decât încap într-un bloc, atunci se creează mai multe blocuri de index, legate între ele sub forma unei liste simplu înlănţuite.

Exemplu. Presupunem că fişierul fis ocupă blocurile `1, 9, 10, 14`. Tabela de index va conţine `4` intrări. Prima intrare va conţine adresa blocului `1`, a `2`-a intrare adresa blocului `9` ş.a.m.d.

Să ne reamintim...

Alocarea contiguă cere ca fiecare fişier să ocupe un set de blocuri consecutive de pe disc. Aici apare fenomenul de **fragmentare**. Opţional se efectuează **compactarea**, în aceleaşi condiţii şi cu aceleaşi metode ca şi cele întâlnite la gestiunea memoriei.

Alocarea înlănţuită rezolvă problemele legate de alocare contiguă. Fiecare fişier este privit ca o listă înlănţuită de blocuri de pe disc. Directorul conţine câte un pointer către primul, respectiv ultimul bloc al fişierului şi fiecare bloc conţine un pointer către următorul bloc al fişierului. O variantă a acestei metode, presupune utilizarea unei tabele de alocare a fişierelor **(FAT-F**ile **A**location **T**able**),** aflată la începutul fiecărui disc logic.

Alocarea indexată presupune că fiecare fişier are o **tabelă de index,** în care se trec în ordine crescătoare adresele tuturor blocurilor ocupate de fişierul respectiv. Cea de-a `i`-a intrare din tabela de index conţine un pointer către cel de-al `i`-lea bloc al fişierului; această tabelă se păstrează într-un bloc separat. În intrarea corespunzătoare fişierului din director, se păstrează această tabelă.

I. Înlocuiți zona punctată cu termenii corespunzători.

1. În cazul alocării contigui dacă presupunem că fişierul conține ..... şi adresa primului bloc este ....., atunci fişierul va ocupa blocurile de adrese .....
2. Alocarea înlănțuită rezolvă problemele legate de alocare contiguă (.....). Fiecare fişier este privit ca ..... de blocuri de pe disc. Directorul conține câte un pointer către ....., respectiv ..... bloc al fişierului şi fiecare bloc conține un pointer către ..... al fişierului.
3. Tabela de alocare a fişierelor este indexată după …… şi fiecare intrare va conține ……., blocului al cărui număr este index, dacă acest bloc este ocupat. Indexului ultimului bloc din fişier, îi corespunde …….. Blocurile neutilizate sunt marcate în tabelă cu …….. Pentru alocarea unui nou bloc unui fişier, se caută ……., şi se înlocuieşte valoarea anterioară de sfârşit de fişier, cu adresa noului bloc, iar valoarea 0 este înlocuită cu ……..
4. În cazul alocării indexate, fiecare fişier are o ……, în care se trec în ordine crescătoare adresele tuturor …….. ocupate de fişierul respectiv.

II. Presupunem că avem un disc cu `32` blocuri (numerotate `1,2,..,32`), în care sunt memorate două fişiere (`fis1, fis2`). Presupunem că, inițial fişierul `fis1` ocupă blocurile `15, 2, 4, 6` iar `fis2` blocurile `16, 3, 1`.

a) Sa se scrie tabela FAT pentru această situație.

b) Care vor fi tabelele FAT, dacă fişierul `fis1` va mai necesita un bloc şi `fis2` două blocuri. Discuție.

M4.U1.8. Evidența spațiului liber de disc

Deoarece spațiul de pe disc este limitat şi se modifică în timp (alocarea de spațiu pentru fişierele nou create sau pentru fişierele în care se adaugă informații şi reutilizarea spațiului datorat ştergerii unor fişiere), sunt necesare metode prin care să se cunoască blocurile libere de pe discuri.

Evidența spațiului printr-un vector de biți. Tabela(vectorul de biți) care ține evidența blocurilor libere, este indexată după numărul de bloc. Dacă un bloc este liber, atunci componenta corespunzătoare a vectorului este setată pe `0`, iar în caz contrar pe `1`.

Exemplu. Presupunem că avem un disc cu `32` de blocuri, numerotate cu `1,..,32`, în care fişierele ocupă blocurile:
`2, 3, 6, 7, 14, 15, 16, 19, 20, 21, 22, 28, 29, 30.`
Atunci vectorul spațiilor libere va fi:
`(0,1,1,0,0,1,1,0,0,0,0,0,0,1,1,1,0,0,1,1,1,1,0,0,0,0,0,1,1,1,0,0 )`

Evidența prin listă înlănțuită. În directorul volumului există un pointer către primul bloc liber. Acest bloc conține un pointer către următorul bloc ş.a.m.d. Ultimul bloc liber conține un pointer nul.

Observație. Ordinea de apariție a blocurilor libere în listă, este dată de acțiunile de cerere şi eliberare a blocurilor efectuate până la momentul respectiv. Lista este organizată pe principiul cozii. Astfel, dacă se cere ocuparea unui bloc, se va lua primul bloc din listă şi se va actualiza valoarea pointerului din director, al doile bloc devenind primul. Eliberarea unui bloc

presupune adăugarea unui nou element în listă, la celălalt cap. Deficiența acestei metode constă în operațiile cu pointeri care sunt foarte costisitoare pe disc.

Exemplu. Presupunem că avem un disc cu `32` de blocuri, numerotate cu `1,..,32`, în care blocurile libere (în ordinea eliberării) sunt: 18, 17, 15, 21, 22, 23, 6, 7, 8, 25, 26, 27. Directorul conține un pointer către blocul 18, care este primul bloc liber, blocul 18 punctează către blocul 17 ş.a.m.d.. Blocul 27 este ultimul din listă, deci va conține valoarea nil în zona alocată pointerului.

Evidența înlănțuită şi indexată constituie o îmbunătățire a metodei anterioare**.** Directorul conține un pointer către primul bloc liber, acesta conține `n` pointeri către alte `n` blocuri libere, dintre care al `n`-lea va conține alți `n` pointeri către alte `n` blocuri libere ş.a.m.d. În acest mod, numărul operațiilor de I/O necesare pentru căutarea de spațiu liber se micşorează în medie de `n-1` ori.

I. Presupunem că avem un disc cu `32` de blocuri, numerotate cu `1,..,32`, în care fişierele ocupă blocurile:
`1, 5, 6, 7, 11, 15, 17, 19, 20, 21, 22, 28, 29, 30.`
Scrieți vectorul de biți corespunzător discului.

M4.U1.9. Test de evaluare a cunoştințelor
Marcați varianta corectă.

1. Un sistem de fişiere de nivel scăzut presupune:
   a) Translatarea articole în fluxuri de octeți, lăsând în seama aplicațiilor structurarea fişierelor ca flux de blocuri. ☐
   b) Deschiderea fişierelor la prima referință la acestea. ☐
   c) Translatarea blocurilor în fluxuri de octeți, lăsând în seama aplicațiilor structurarea fişierelor ca flux de articole. ☐
   d) Înhiderea fişierelor la ultima referință la acestea. ☐
2. Numim index de articol**:**
   a) Numărul de ordine al articolului. ☐
   b) Un atribut cu proprietatea că pentru oricare două articole diferite ale fişierului, valorile atributului sunt diferite. ☐
   c) Primul atribut al unei înregistrări. ☐
   d) Un atribut cu proprietatea că pentru oricare două articole diferite ale fişierului, valorile atributului sunt egale. ☐
3. Accesarea secvențială a unui fişier presupune parcurgerea înregistrărilor în ordinea dată de:
   a) Succesiunea temporală a scrierii înregistrărilor din fişier. ☐
   b) Ordinea crescătoare a adreselor înregistrărilor din fişier. ☐
   c) Ordinea descrescătoare a adreselor înregistrărilor din fişier. ☐
   d) Ordinea lungimii înregistrărilor. ☐
4. Unitatea de schimb între suportul fişierului şi memoria internă este:
   a) Înregistrarea. ☐
   b) Articolul. ☐
   c) Blocul logic. ☐
   d) Întregul fişier. ☐
5. Atât pentru un fişier nou creat, cât şi unul existent, operația de deschidere face:
   a) Legătura dintre identificatorul logic, utilizat de program şi descriptorul de fişier aflat pe disc ☐
   b) Legătura dintre identificatorul logic, utilizat de program şi directorul tată al fişierului. ☐
   c) Legătura dintre identificatorul logic, utilizat de program şi adresa de pe disc a fişierului. ☐
   d) Legătura dintre identificatorul logic, utilizat de program şi tabela fişierelor deschise. ☐

6. Care operație se execută la închiderea fișierului, pentru fișierele nou create și care trebuie reținute:

a) Se golește tamponul adică ultimele informații existente în zonele tampon sunt transferate pe periferic. ☐

b). Se golește tamponul adică ultimele informații existente în zonele tampon sunt afișate. ☐

c). Se golește tamponul adică ultimele informații existente în zonele tampon sunt șterse. ☐

d) Se golește tamponul adică ultimele informații existente în zonele tampon sunt transferate în memoria virtuală . ☐

7. Organizarea indexat-secvențială urmărește:

a) Minimizarea numărului de comparații necesare accesării unei înregistrări dintr-un fișier. ☐

b) Optimizarea procesului de căutare a fișierelor pe un disc. ☐

c) Optimizarea procesului de căutare a fișierelor de pe un director. ☐

d) Optimizarea alocării de spațiu pe disc. ☐

8. În cadrul tabelei de indecși, pentru fiecare pagină se memorează:

a). Adresa de pe disc a paginii și dimensiunea paginii. ☐

b). Adresa de pe disc a paginii și valoarea minimă a indecșilor din pagină . ☐

c). Adresa de pe disc a paginii și adresa primului articol din pagină. ☐

d) Adresa de pe disc a paginii și valoarea maximă a indecșilor din pagină. ☐

9. Clasa de sinonimie se referă la:

a) Organizarea secvențială a fișierelor. ☐

b) Organizarea selectivă a fișierelor. ☐

c) Organizarea indexat secvențială a fișierelor. ☐

d) Organizarea aleatoare a fișierelor. ☐

10. Directoarele cu structură de arbore au fost introduse pentru:

a) Minimizarea numărului de comparații necesare accesării unei înregistrări dintr-un fișier. ☐

b) A înlocui căutarea secvențială a fișierelor pe un disc cu căutarea arborescentă. ☐

c) Optimizarea alocării de spațiu pe disc. ☐

d) A înlocui căutarea secvențială a înregistrărilor dintr-un fișier pe un disc cu căutarea arborescentă. ☐

11. Directoarele sub formă de structură de graf permit:

a) Partajarea unui disc ☐

b) Accesarea mai rapidă a înregistrărilor unui fișier. ☐

c) Partajarea unui fișier.. ☐

d) Evidența utilizatorilor unui fișier. ☐

12 În cadrul cărui tip de alocare a spațiului pentru fișiere pe disc, în descriptorul de fișier din director, trebuie să se memoreze adresa de început și lungimea zonei alocate:

a) Contiguă. ☐

b) Secvențială. ☐

c) Înlănțuită. ☐

d) Indexat secvențială. ☐

M4.U1.10 Rezumat

Fişierele reprezintă mijloacele principale prin care se stochează şi manipulează cantităţi mari de informaţie pe un sistem de calcul. Fişierele pe care le poate gestiona un sistem de operare pot fi fluxuri de octeţi sau fluxuri de înregistrări, care la rândul lor pot avea o structură liniară sau arborescentă. În prima situaţie, sistemul de operare oferă mijloace generice şi flexibile pentru manipularea datelor, aplicaţiile fiind cele care oferă posibilităţi de structurare a datelor. A doua situaţie, presupune un sistem de operare destul de voluminos, ale cărui facilităţi suplimentare nu sunt întotdeauna necesare.

Sistemul de fişiere oferă facilităţi de transformare a fişierelor din flux de biţi/articole în succesiuni de blocuri, formă sub care sunt manipulate la nivel de disc şi care permite o utilizare mai eficientă a operaţiilor de citire/scriere.

Modurile de accesare a informaţiilor din fişier a evoluat în timp, de la fişier cu acces secvenţial la fişiere cu acces indexat-secvenţial sau selectiv, sau direct, în scopul diminuării timpului de căutare a informaţiei într-un fişier.

Pentru o regăsire mai rapidă a fişierelor de pe disc şi pentru o gestionare a lor mai eficientă a apărut conceptul de director cu structură arborescentă sau graf aciclic.

Evidenţa spaţiului liber de pe disc şi a blocurilor utilizate de către fişiere sunt alte sarcini pe care le rezolvă sistemul de fişiere.

.

Modulul 5. Administrarea perifericelor

Unitatea de învățare M5U1. Administrarea perifericelor

Cuprins

Introducere.

Una dintre sarcinile principale ale sistemului de operare este controlul echipamentelor de intrare-ieşire (I/O). El trebuie să transmită comenzi către acestea, să trateze eventuale întreruperi sau erori. De asemenea, sistemul de operare trebuie să furnizeze o interfață simplă, uşor de utilizat de către utilizator. În acest sens, controlul perifericelor unui calculator este o preocupare importantă a proiectanților de sisteme de operare.

O altă problemă care trebuie rezolvată este creşterea vitezei de lucru a sistemelor de calcul, printr-o bună gestionare a operațiilor de I/O, datorită colaborării eficiente dintre CPU şi subsistemul de administrare a unităților de I/O, care este o componentă a sistemului de operare.

Un alt aspect este diversitatea din punct de vedere hardware a perifericelor, atât ca funcții cât şi ca viteză de lucru; astfel, este necesară o interfață software care să poată gestiona aceste diferențe, care să omogenizeze utilizarea lor de către aplicațiile software.

Obiectivele unității de învățare.

La sfârşitul acestei unități de învățare studenții vor fi capabili:

- să explice legătura dintre driver, ca şi componentă a SO şi controllerul de unitate, ca interfață hardware;
- să explice modalitățile prin care CPU participă la efectuarea operațiilor de I/O, precum şi rolul sistemului de operare ca interfață între programe şi periferice;
- să explice conceptul de utilizare a zonelor tampon şi se evidențiază implicațiile lui asupra creşteri performanțelor în executarea operațiilor de I/O;

să explice conceptele de acces direct la memoria internă al perifericelor şi proiecția în memorie a I/O, specifice calculatorelor moderne şi implicațiile lor asupra creşterii vitezei de prelucrare.

Durata medie de parcurgere a unității de învățare este de 3 ore.

M5.U1.3. Organizarea sistemului de I/O

Unitățile de I/O(perifericele) sunt folosite pentru a plasa date în memoria primară şi pentru a stoca cantități mari de informații pentru o perioadă lungă de timp.
Ele pot fi:
- Unități de stocare (unități bloc): dispozitivele bloc stochează informația sub forma unor blocuri de dimensiune fixă, care pot fi accesate individual, prin intermediul unei adrese.

Exemplu. Discurile

- Unități caracter: lucrează cu şiruri de caractere care nu sunt structurate ca blocuri.

Exemplu. Tastatura, mouse-ul, display-ul terminalului

- Unități de comunicație:portul serial conectat la un modem sau o interfață la rețea.

Exemplu. Portul serial conectat la un modem sau o interfață la rețea

În general, unitățile de I/O sunt formate din:
- o componenta mecanică - echipamentul propriu-zis;
- o componenta electronică, numită **controller de echipament**; la calculatoarele personale acesta este prezent uneori sub forma unei plăci cu circuit electronic ce poate fi introdusă într-un slot de expandare; de multe ori, controloarele au conectori la care pot fi conectate prin cablu echipamentele şi mai multe echipamente de acelaşi fel pot fi conectate la un acelasi controlor.

Un sistem de I/O trebuie să răspundă la cerințele următoare:
►Să poată recunoaşte caractere, indiferent de codul în care acestea sunt reprezentate.
►Să realizeze independența față de dispozitvele periferice a programelor, în sensul că acestea nu trebuie să sufere modificări importante atunci când tipul dispozitivului periferic utilizat este altul decât cel prevăzut inițial. Acest lucru este posibil prin furnizarea unor operații a căror sintaxă şi semantică să fie aceeaşi pentru o clasă cât mai mare de unități de I/O, precum şi denumirea uniformă a perifericelor, lucru realizat prin asocierea de fişiere acestor dispozitive.

►Realizarea operațiilor într-un timp cât mai mic, adică găsirea unor metode care să anihileze diferențele între viteza de operare a CPU şi cea a perifericelor.

Figura 5.1.1 ilustrează componentele implicate în realizarea operațiilor de I/O.

Exemplu. Să aloce/elibereze o unitate, să pună sub/scoată de sub tensiune un disc, să modifice imaginea pe un anumit display etc.

API (Application Programming Interface) furnizează un set de funcții pe care un programator le poate apela pentru a utiliza o unitate. Proiectantul SO trebuie să furnizeze şi alte funcții API care să permită unui proces să efectueze şi alte operații.

Componenta sistemului de operare care manipulează dispozitivele de I/O este formată din driverele de unitate. Driverul are două sarcini principale:
►Implementează o interfață API la care procesul are acces.

►Furnizează operații dependente de unitate, corespunzătoare funcțiilor implementate de API.

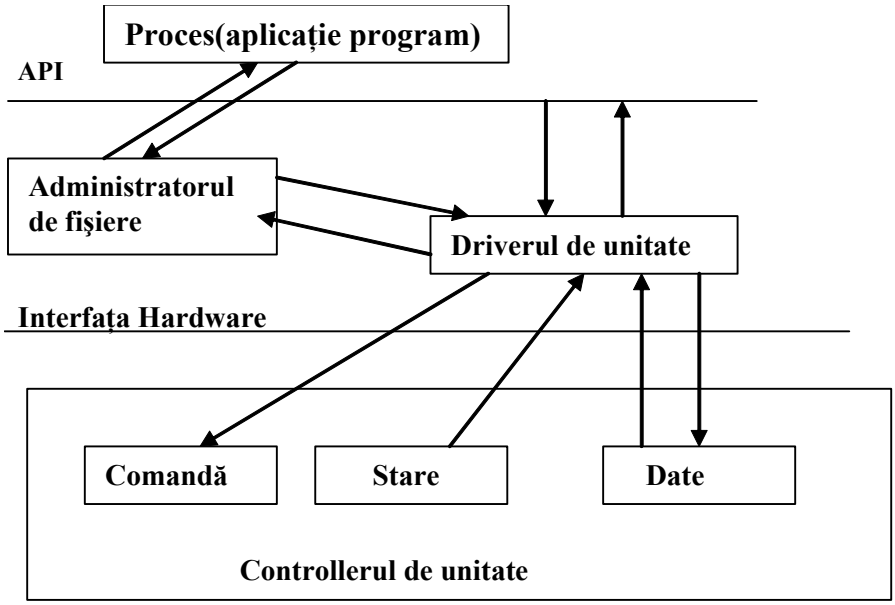


Figura 5.1.1. Componentele implicate în realizarea operațiilor de I/O

Fiecare driver de unitate furnizează funcții care să permită unui program să citească/scrie de pe/pe o unitate. Funcțiile de **citire/scriere** diferă din punct de vedere sintactic în funcție de tipul unității(caracter sau bloc). Semantica fiecărei funcții depinde de modul de accesare a unității (secvențială sau directă).

Exemplu. Unitățile de tip bloc conțin comenzi de tipul `read()`, `write()` și `seek()`, pentru a citi, scrie de pe/pe unitatea respectivă și pentru a accesa direct un bloc de informații. În cazul unităților de tip caracter, cum este cazul tastaturii sau ecranului unui monitor, apelurile de sistem ale acesteia permit comenzi de tipul `get()` sau `put()`, pentru citirea/scrierea caracter cu caracter.

Fiecare sistem de operare definește o arhitectură pentru sistemul său de administrare a unităților. Nu există un model universal, dar fiecare din aceste sisteme conține o API și o interfață între driver și nucleul sistemului de operare.

Administratorul unității trebuie să urmărească starea unității, când este liberă, când este folosită și care proces o folosește. Acesta poate întreține, în plus față de informațiile păstrate în tabela de stare a unității, un descriptor de unitate care specifică alte caracteristici ale unității, determinate de proiectantul de software. Deci interfețele driverelor includ funcțiile de deschidere(open) și închidere(close) pentru a permite administratorului să înceapă/termine utilizarea unității de către un proces. Comanda **open** alocă perifericul și inițializează tabelele și unitatea pentru utilizare iar **close** șterge informațiile scrise în tabele, eliberând astfel unitatea.

Interfața cu nucleul. Atunci când interacționează cu controllerul de unitate, driverul de unitate execută instrucțiuni privilegiate Deci, driverul trebuie să fie executat ca parte a sistemului de operare și nu ca parte a programului.

Driverul trebuie să fie capabil să citească/scrie informații din/în spațiile de adrese ale diferitelor procese, deoarece aceeași unitate poate fi folosită de către diferite procese.

În sistemele mai vechi, driverul este încorporat în sistemului de operare prin modificarea codului sursă al sistemului de operare şi, apoi recompilarea acestuia. Deci, proprietarul sistemului de calcul trebuie să deţină sursele sistemului de operare şi să cunoască modul cum trebuie realizată compilarea lui. În condiţiile existenţei sistemelor deschise, acest mod de lucru devine inacceptabil.

Sistemele de operare moderne simplifică instalarea driverelor prin utilizarea **driverelor de unitate reconfigurabile**. Astfel de sisteme permit adăugarea unui driver de unitate fără a fi necesară recompilarea sistemului de operare, fiind necesară numai **reconfigurarea** sistemului printr-un set de operaţii oferite chiar de către sistemul de operare. Pentru aceasta, la nivelul fiecărei unităţi, există o tabelă care conţine pointeri către modulele sursă ale funcţiilor interfeţei. Un driver de unitate reconfigurabil trebuie să fie standardizat, adică să ofere aceeaşi interfaţă.

Deoarece driverul de unitate este adăugat nucleului după ce acesta a fost compilat, nucleul trebuie să furnizeze o interfaţă care să permită driverului de unitate să aloce spaţiu pentru buffere, să manipuleze tabele ale nucleului etc.

Interfaţa cu CPU. Perifericele şi CPU sunt componente distincte care sunt capabile să lucreze independent. Administratorul unităţii trebuie să furnizeze metodele prin care să se coordoneze execuţia procesului şi operaţiile de I/O.

În efectuarea operaţiilor de I/O sunt implicate trei componente: controllerul, driverul şi apeluri de funcţii de I/O conţinute în programul în execuţie. Între acestea trebuie să existe o bună coordonare. Deoarece controllerul este o componentă hardware, el ori execută o operatie de I/O, ori este în aşteptare activă pentru a primi în registrul de comenzi codul unei operaţii. Driverul unităţii este o componentă software a sistemului de operare invocată de către procesul în execuţie. Fiecare funcţie a driverului este o procedură apelată de către procesul în execuţie. Toate aceste componente software implicate în lucrul cu controllerul unităţii sunt nişte procese, gestionate de către CPU pe baza algoritmilor de planificare studiaţi.

Să ne reamintim...
Unităţile de I/O(perifericele) sunt folosite pentru a plasa date în memoria primară şi pentru a stoca cantităţi mari de informaţii pentru o perioadă lungă de timp.Ele pot fi: unităţi de stocare (unităţi bloc), unităţi caracter, unităţi de comunicaţie.

În general, unităţile de I/O sunt formate dintr-o componenta mecanică şi o componenta electronică, numită controller de echipament.

API (Application Programming Interface) furnizează un set de funcţii pe care un programator le poate apela pentru a utiliza o unitate. Componenta SO care manipulează dispozitivele de I/O este formată din driverele de unitate. Fiecare driver de unitate furnizează funcţii care să permită unui program să citească/scrie de pe/pe o unitate.

Înlocuiţi zona punctată cu termenii corespunzători.

1. Unităţile de I/O(perifericele) sunt folosite pentru ..... şi pentru a stoca .....pentru .....
2. Dispozitivele bloc stochează informaţia sub forma unor ....., care pot fi accesate ....., prin intermediul unei .....
3. Un sistem de I/O trebuie să poată recunoaşte ....., indiferent de ..... acestea sunt reprezentate.
4. Un sistem de I/O trebuie să realizeze ..... faţă de dispozitvele periferice a programelor.

5. Realizarea operațiilor de I/O trebuie să se facă ....., adică să se găsescă care să anihileze diferențele între ..... şi cea a .....
6. Driverul implementează o …… la care procesul are acces.
7. Driverul furnizează operații ……., corespunzătoare funcțiilor …….
8. Driverul trebuie să fie executat ca parte a ..... şi nu ca parte a programului.

M5.U1.4. Controller de unitate

Fiecare unitate foloseşte **controllerul de unitate** pentru a o conecta la adresele calculatorului şi la magistrala de date. Controller-ul oferă un set de componente fizice pe care instrucțiunile CPU le pot manipula pentru a efectua operații de I/O. Ca şi construcție, controller-ele diferă, dar fiecare oferă aceeaşi interfață de bază.

SO ascunde aceste detalii de funcționare ale controller-lor, oferind programatorilor funcții abstracte pentru accesul la o unitate, scrierea/citirea de informații etc. În timpul lucrului, orice unitate ar trebui monitorizată de către CPU. Acest lucru este făcut de către controller-ul de unitate.

Interfața între controller şi unitate caută să rezolve problema compatibilității între unități fabricate de diverşi producători. O astfel de interfață este SCSI (Small Computer Serial Interface). Interfața între magistrală şi controller este importantă pentru cei care se ocupă cu adăugarea unei noi unități la arhitectura existentă, pentru ca unitatea adăugată să poată lucra împreună cu celelalte componente.

Interfața între componenta software şi controller se realizează prin intermediul **regiştrilor** controller-ului. Această interfață defineşte modul cum componenta soft manipulează controller-ul pentru a executa operațiile de I/O. Figura 5.1.2 ilustrează interfața software cu un controller.

Unitățile conțin două fanioane în registrul lor de stare: **busy** şi **done**. Dacă ambele fanioane sunt setate pe `0`(`false`), atunci componenta software poate plasa o comandă în registrul de comenzi pentru a activa unitatea. După ce componenta software a pus informații într-unul sau mai mulți regiştri de date, pentru o operație de ieşire, unitatea este disponibilă pentru a fi utilizată. Prezența unei noi comenzi de I/O are ca efect poziționarea fanionului **busy** pe `true`, începerea efectuării operației şi datele din registrul de date sunt scrise pe unitate.

O operație de citire se efectuează dual. Procesul poate detecta starea operației verificând registrul de stare. Când operația de I/O a fost terminată(cu succes sau eşec), controller-ul şterge fanionul **busy** şi setează fanionul **done**. Odată cu terminarea unei operații **read**, respectiv **write** datele sunt copiate de pe unitate în registrul de date, respectiv invers. Dacă după o operație de scriere ambele fanioane ale unității au fost setate pe false, scrierea de noi informații în registrul de date va fi sigură. Când procesul citeşte date din controller, acesta şterge registrul **done**, adică controller-ul este din nou gata de utilizare. Dacă operația se termină eronat, câmpul cod de eroare va fi setat. Controller-ele pot conține o mică zonă de memorie (buffer), pentru a stoca datele după ce au fost citite, înainte de a fi transmise CPU pentru prelucrare. Invers, în memorie pot fi păstrate datele care aşteaptă să fie scrise.

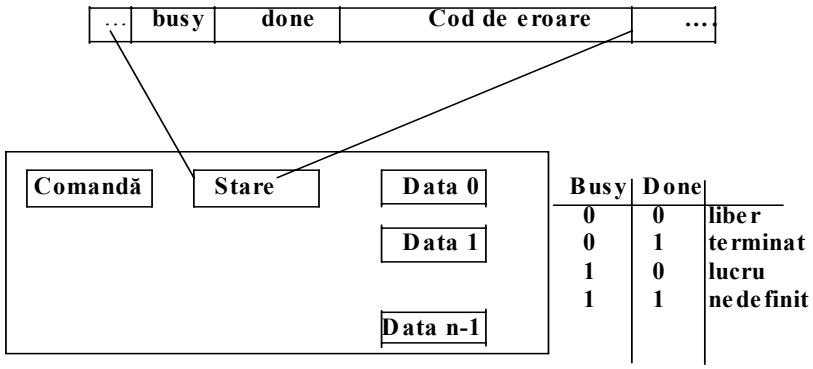


Figura 5.1.2. Interfața controllerului de unitate

Prezența unităților, controllerelor şi driverelor(figura 5.1.3) permite ca două părți ale aceluiaşi program să poată lucra în paralel, o parte folosind CPU iar altă parte folosind unitățile. Driverul corespunzător unei anumite interfețe poate fi relativ complex, în sensul că trebuie setați mai mulți parametri înainte de pornirea controllerului şi trebuie verificate mai multe aspecte ale stării atunci când operația a fost terminată. Driverele pentru diferite unități implementează o interfață similară, deci programatorii care lucrează cu limbaje de nivel înalt nu trebuie să fie preocupați de detaliile unei anumite unități. Scopul este de a simplifica cât mai mult posibil interfața cu unitățile.

Driverele de unitate sunt entități ale SO care controlează paralelismul, între activitățile efectuate de CPU şi operațiile de I/O. O unitate poate fi activată prin intermediul driverului corespunzător ei şi procesul care a generat activarea poate fi executat în continuare, în paralele cu operațiile asupra unității. Problema care se pune, este de a semnala procesului care a generat execuția driverului(a activat unitatea), când se termină operația de I/O respectivă.

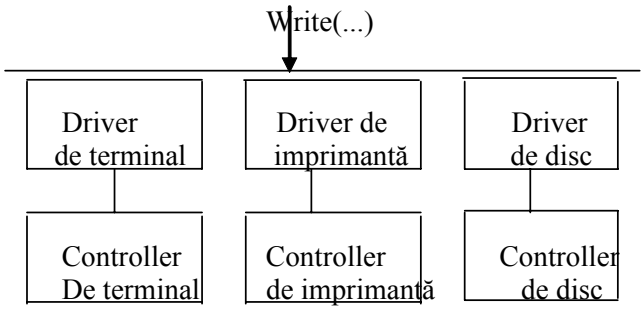


Figura 5.1.2. Corespondența dintre operații, drivere şi controllere de unitate

Observații.

- Se reduce cantitatea de informații pe care procesul (aplicația) trebuie să le cunoască pentru a utiliza unitatea respectivă.
- Fiecare driver este specific unei anumite unități, fiind capabil să furnizeze controllerului comenzile corespunzătoare, să interpreteze corect conținutul registrului de stare al controllerului şi să transforme date la/de la regiştrii de date ai controllerului, în confomitate cu cerințele operației respective.
- Un proces care foloseşte o unitate, îi transmite acesteia comenzi şi schimbă date cu driverul de administrare al unității respective.
- Driverele sunt componente ale administratorului de unitate folosite de către un proces pentru a cere efectuarea unei operații de I/O. Când procesul face o cerere către driverul de unitate, acesta o translatează într-o acțiune dependentă de controller, pentru a realiza interacțiunea cu

unitatea respectivă. După ce unitatea îşi începe lucrul, driverul fie testează periodic controllerul pentru a detecta sfârşitul operaţiei, fie plasează informaţii în tabela de stare a unităţii pentru a pregăti lansarea unei întreruperi.

Să ne reamintim...

Fiecare unitate foloseşte **controllerul de unitate** pentru a o conecta la adresele calculatorului şi la magistrala de date. Controller-ul oferă un set de componente fizice pe care instrucţiunile CPU le pot manipula pentru a efectua operaţii de I/O. SO ascunde aceste detalii de funcţionare ale controller-lor, oferind programatorilor funcţii abstracte pentru accesul la o unitate, scrierea/citirea de informaţii.

Interfaţa între controller şi unitate caută să rezolve problema compatibilităţii între unităţi fabricate de diverşi producători. Interfaţa între componenta software şi controller se realizează prin intermediul **regiştrilor** controller-ului. Prezenţa unităţilor, controllerelor şi driverelor permite ca două părţi ale aceluiaşi program să poată lucra în paralel, o parte folosind CPU iar altă parte folosind unităţile.

Înlocuiţi zona punctată cu termenii corespunzători.

1. Fiecare unitate foloseşte **.....** pentru a o conecta la ..... şi la .....
2. Interfaţa între controller şi unitate caută să rezolve problema ..... între unităţi fabricate de diverşi producători.
3. Interfaţa între componenta software şi controller se realizează prin intermediul **.....** controller-ului.
4. Dacă fanioanele **busy** şi **done** ale controllerului sunt ....., atunci componenta software poate plasa o comandă în .....pentru a activa unitatea.
5. Driverele de unitate sunt ..... care controlează paralelismul, între activităţile efectuate de .....şi .....

M5.U1.5. Metode clasice de administrare a I/O

Administrarea directă a I/O cu testare periodică(polling) se referă la efectuarea operaţiilor de I/O cu participarea CPU, care este responsabilă de transferul datelor între memoria internă a SC şi regiştrii de date ai controllerului. CPU porneşte unitatea şi testează periodic registrul de stare al acesteia, pentru a determina momentul când operaţia s-a terminat.

În cazul efectuării unei operaţii de **citire**, această metodă constă în următorii paşi:
1. Procesul în execuţie(corespunzator unei aplicaţii) cere efectuarea unei operaţii **read**.
2. Driverul de unitate(specificat în comanda de citire) testează registrul de stare pentru a vedea dacă unitatea este liberă. Dacă unitatea este ocupată, driverul asteaptă până când aceasta se eliberează (aşteptare activă), după care se trece la pasul următor.
3. Driverul pune codul comenzii de intrare în **registrul comandă al controllerului**, pornind astfel unitatea.
4. Driverul citeşte în mod repetat **registrul de stare**, atâta timp cât este în aşteptarea terminării operaţiei.
5. Driverul copiază conţinutul **regiştrilor de date** ai controlerului în spaţiul de memorie al procesului utilizator.
6. După terminarea operaţiei, se revine la următoarea instrucţiune din program.

Paşii necesari pentru a executa o operaţie de **scriere** sunt:
1. Procesul în execuţie(aplicaţia) cere execuţia unei operaţii **write**.

2. Driverul unității specificate interoghează **registrul de stare** al acesteia, pentru a vedea dacă unitatea este liberă. Dacă unitatea este ocupată, driverul aşteaptă până când aceasta devine liberă, după care se trece la pasul următor.
3. Driverul transferă date din **spațiul de memorie** al procesului în **regiştrii de date** ai controllerului.
4. Driverul scrie un cod de comandă de ieşire în **registrul comandă**, pornind astfel unitatea.
5. Driverul citeşte în mod repetat **registrul de stare**, atâta timp cât aşteaptă terminarea operației.
6. După terminarea operației, se revine la următoarea instrucțiune din program.

Observații.
- Fiecare operație de I/O necesită o coordonare a componentelor software şi hardware implicate. Prin metoda anterioară, această coordonare este realizată prin încapsularea părții soft a interacțiunii (driverul), cu controllerul unității respective.
- Această metodă duce la încărcarea CPU cu sarcini suplimentare, şi anume verificarea periodică a stării controllerului.

Administrarea operațiilor de I/O orientată pe întreruperi. Din punct de vedere funcțional, operația de I/O este realizată prin cooperarea mai multor componente hard şi soft, şi anume:
►Driverul de unitate, care declanşează operația.
►Tabela de stare a unității.
►Administratorul (handler-ul) întreruperilor.
►Administratorul unității.

Pentru a realiza o operație de intrare sunt efectuați următorii paşi:
1. Procesul cere efectuarea unei operații **read**.
2. Driverul interoghează **registrul de stare** pentru a vedea dacă unitatea este liberă. Dacă este ocupată, driverul aşteaptă ca aceasta să devină liberă (aşteptare activă), după care se trece la pasul următor.
3. Driverul pune codul operației în **registrul comandă** al controllerului, pornind astfel unitatea.
4. Driverul îşi termină sarcina, salvează informația cu privire la operația pe care a început-o în intrarea din tabela de stare a unităților, corespunzătoare unității folosite; aceasta conține **adresa de retur** din procesul care a lansat operația şi alți parametri speciali ai operației de I/O. Astfel, din acest moment, CPU poate fi folosită de către alte programe, deci **administratorul unității** transmite un mesaj corespunzător componentei de planificare a administratorului de procese.
5. Unitatea termină de efectuat operația şi atunci lansează o **întrerupere** către CPU; de rezolvarea acesteia se va ocupa mai departe handler-ul de întreruperi.
6. Administratorul(handler-ul) întreruperilor determină care unitate a cauzat întreruperea, după care face un salt la administratorul (handler-ul) acelei unități.
7. Administratorul unității găseşte informațiile de stare ale unității de I/O în tabela de stare a unităților; sunt utilizate pentru terminarea operației.
8. Administratorul unităților copiază conținutul regiştrilor de date ai controllerului în spațiul de memorie al procesului.
9. Administratorul unității transferă controlul procesului apelant.

Observații.
- Operația de ieşire se desfăşoară similar. Din punctul de vedere al procesului în execuție, această activitate are aceeaşi semnificație ca oricare apel de procedură.
- Când este folosită o instrucțiune read într-un program, aceasta trebuie să se termine înainte ca următoarea instrucțiune să fie executată. Dacă CPU a început să execute o instrucțuine în care este implicată o variabilă în care s-a efectuat citirea, după ce driverul de unitate a început

să execute operația de citire, dar înainte ca ea să se termine, atunci instrucțiunea respectivă va fi executată folosind o valoare veche a variabilei respective, nu cea care este în curs de citire de la unitate. Pentru a preveni aceasta situație SO blochează explicit procesul care a inițiat apelul read. SO aşteaptă ca procesul să termine operația de I/O înainte de a executa instrucțiunea următoare.
- Deşi procesul însuşi nu este capabil să obțină avantajele desfăşurării în paralel a activității CPU şi a operațiilor de I/O, totuşi SO poate comuta CPU la un alt proces, oricând un proces cere efectuarea unei operatii de I/O. Astfel, perfomanțele generale ale sistemului pot fi îmbunătățite prin lucrul în paralel al CPU, cu efectuarea unei operații de I/O. Când operația de I/O s-a terminat, procesul care a inițiat operația va fi replanificat. Mecanismul de planificarea al proceselor va interveni atât în momentul terminării operației de I/O, cât şi atunci când se realizează o serializare a operației de I/O şi a instrucțiunii imediat următoare acesteia.

Administrarea directă a I/O cu testare periodică(polling) se referă la efectuarea operațiilor de I/O cu participarea CPU, care este responsabilă de transferul datelor între memoria internă a SC şi regiştrii de date ai controllerului. CPU porneşte unitatea şi testează periodic registrul de stare al acesteia, pentru a determina momentul când operația s-a terminat.

Administrarea operațiilor de I/O orientată pe întreruperi este realizată prin cooperarea mai multor componente hard şi soft, şi anume:driverul de unitate, care declanşează operația; tabela de stare a unității; administratorul (handler-ul) întreruperilor şi administratorul unității.

Înlocuiți zona punctată cu termenii corespunzători.

1. În cazul administrarării directe a I/O cu testare periodică, pentru a lansa o operație de citire ..... cere efectuarea unei operații .....
2. În cazul administrarării directe a I/O cu testare periodică, driverul de unitate, specificat în ..... testează ..... pentru a vedea dacă unitatea este liberă.
3. În cazul administrarării directe a I/O cu testare periodică, în cazul operației de citire driverul copiază conținutul **.....** ai ..... în ..... al procesului utilizator.
4. În cazul administrarării directe a I/O cu testare periodică, , în cazul operației de citire driverul transferă date din **.....** al procesului în ..... ai controllerului.
5. În cazul administrarării operațiilor de I/O orientată pe întreruperi, în cazul operației de citire, driverul interoghează ..... pentru a vedea dacă .....este liberă.
6. În cazul administrarării operațiilor de I/O orientată pe întreruperi, când unitatea termină de efectuat operația lansează ..... către .....
7. În cazul administrarării operațiilor de I/O orientată pe întreruperi, administratorul unității găseşte informațiile de stare ale unității de I/O ....., care sunt utilizate pentru .....operației.

M5.U1.6. Utilizarea întreruperilor

Execuția unui program. CPU extrage şi execută instrucțiunile cod maşină ale procesului încărcat în memoria internă. În acest sens, CPU conține:
►o componentă care extrage o instrucțiune memorată într-o locație de memorie;
►o componentă care decodifică instrucțiunea;
►o componentă care se ocupă de execuția instrucțiunii, împreună cu alte componente ale SC.

Registrul **contor de program** PC (Program Counter) conține adresa de memorie a instrucțiunii care urmează să fie executate iar **registrul instrucțiune** IR (Instrucțion Register) o copie a instrucțiunii în curs de prelucrare.

Modul de operare al CPU se desfășoară după următorul **algoritm**:

```
PC = <Adresa de început a procesului> ;
IR = M[PC];
HaltFlag = CLEAR;
While (HaltFlag nu este setat în timpul execuției )
{
     PC = PC+1;
     Execute(IR);
     IR = M[PC];
}
```

Vectorul `M` identifică locațiile de memorie internă ale SC, adică `M[i]` reprezintă locația de memorie `i`. Când un proces urmează să fie executat, PC este încărcat cu adresa primei sale instrucțiuni, iar în IR este încărcat codul instrucțiunii respective. După terminarea execuției unei instrucțiuni, PC este incrementat, adică se trece la execuția următoarei instrucțiuni, ș.a.m.d. Fanionul HaltFlag este testat înainte de a se trece la execuția instrucțiunii, pentru a se verifica dacă CPU este oprită. **Execute** reprezintă procedura care corespunde instrucțiunii extrase.

Testarea fanioanele **busy-done** de către CPU prin intermediul driverelor de unitate duce la utilizarea ineficientă a CPU. Cea mai bună soluție pentru desfăşurarea paralelă a operațiilor de I/O şi a activității CPU, este ca atunci când operația de I/O se termină, să se semanaleze acest lucru procesorului, lucru care se realizează prin intermediul **întreruperilor**. În partea de hardware este încorporat un fanion **InReq** (**In**terrupt **Req**uest) şi unitatea de control este astfel concepută încât să poată verifica fanionul IR de fiecare dată când extrage o nouă instrucțiune. Algoritmul de extragere şi execuție al instrucțiunilor procesului atunci când SC foloseşte întreruperi este:

```
While (HaltFlag nu este setat)
{
IR = M[PC];
PC = PC + 1;
Execute(IR);
If (InReq) // Are loc întrerupere procesului curernt
     {M[0] = PC;//Salvează val. curentă a lui PC la
     adresa 0
     PC=M[1]}
}
```

Partea de hardware conectează toate fanioanele done ale unităților la fanionul **InReq** prin efectuarea unei operații SAU. Oricând este setat un fanion done al unui controller, fanionul InReq va fi şi el setat. CPU va fi atenționată despre terminarea operației de I/O în intervalul de timp cât execută o instrucțiune, urmând ca după terminarea acesteia să trateze întreruperea respectivă. Semnalul de la componenta hardware, la cea software (întreruperea) are ca efect oprirea execuției secvenței de instrucțiuni ale procesului respectiv de către processor (adresate de către PC) şi saltul la o nouă secvență de instrucțiuni, a cărei adresă este conținută la o adresă de memorie(de exemplu `M[1]`). Înainte de a se efectua saltul, valoarea lui PC (adresa instrucțiunii care urmează să fie executată) împreună cu starea procesului sunt salvate prin intermediul hardwarelui. Adresa componentei SO care manipulează întreruperile (IH - Interrupt Handler) este memorată în locația de memorie `M[1]`.

Când IH îşi începe execuţia, regiştrii CPU conţin valori care urmează să fie folosite de către procesul care a fost întrerupt. IH trebuie să execute o comutare de context pentru a salva conţinutul tuturor regiştrilor de stare şi generali utilizaţi de proces şi să-şi pună propriile valori în regiştrii CPU pentru a putea realiza terminarea operaţiei de I/O corespunzătoare întreruperii lansate. Algoritmul după care lucrează IH este:

```
SaveProcessorState();
For (i=0;i<Numărul de unităţi; i++)
   If (Device[i].done == 1) goto device_handler(i);
   If (InReq && InterruptEnabled)
     {
          DisableInterrupts();
          M[0] = PC;
          PC = M[1]
     }
```

IH testează valorile fanioanelor done ale controllerelor de unitate pentru a determina care unitate a terminat operaţia de I/O(a cauzat întreruperea). Dacă două sau mai multe unităţi termină operaţia de I/O în timpul execuţiei aceleiaşi instrucţiuni cod maşină, atunci, se va detecta numai prima dintre ele. Odată ce cauza întreruperii a fost determinată, IH face un salt la codul întreruperii corespunzător unităţii a cărei terminare a operaţiei de I/O trebuie tratată.
Dacă o întrerupere este declanşată în momentul când IH este în timpul execuţiei, adică tratează o întrerupere apărută anterior, aceasta va aştepta până la terminarea întrerup. ant. Dacă IH ar trata ambele întreruperi în paralel, ar exista posibilitatea ca apariţia unor erori în execuţia uneia dintre întruperi să ducă la eşecul execuţiei ambelor întreruperi.

Orice instrucţiune care setează fanionul **HaltFlag** al CPU este o instrucţiune privilegiată, deoarece poziţionarea pe TRUE a lui HaltFlag are ca efect oprirea execuţiei oricărei instrucţiuni. Dacă CPU lucrează în mod utilizator şi un proces doreşte să înceapă execuţia în modul supervizor, mediul trebuie să se asigure că atunci când modul de lucru este comutat din modul utilizator în cel supervizor, CPU va începe să execute instrucţiunile nucleului sistemului. Modul supervizor este setat de către instrucţiune cod maşină trap de forma:

```
trap argument
```

Mai întâi CPU comută din modul utilizator în supervizor, după care va sări la o adresă care conţine adresa acelei secvenţe de instrucţiuni(întreruperi) corespunzătoare argumentului instrucţiunii trap

Să ne reamintim...
Pentru execuţia unui program, CPU extrage şi execută instrucţiunile cod maşină ale procesului încărcat în memoria internă. În acest sens, CPU conţine:o componentă care extrage o instrucţiune memorată într-o locaţie de memorie;o componentă care decodifică instrucţiunea; o componentă care se ocupă de execuţia instrucţiunii, împreună cu alte componente ale SC. Regiştrii contor de program, respectiv **registrul instrucţiune** , conţin adresa de memorie, respectiv o copie a instrucţiunii în curs de prelucrare.

Testarea fanioanele busy-done de către CPU prin intermediul driverelor de unitate duce la utilizarea ineficientă a CPU. Cea mai bună soluţie pentru desfăşurarea paralelă a operaţiilor de I/O şi a activităţii CPU, este ca atunci când operaţia de I/O se termină, să se semanaleze acest lucru procesorului, lucru care se realizează prin intermediul **întreruperilor**. În partea de hardware este încorporat un fanion **InReq** (**In**terrupt **Req**uest) şi unitatea de control este astfel concepută încât să poată verifica fanionul IR de fiecare dată când extrage o nouă instrucţiune.

Înlocuiți zona punctată cu termenii corespunzători.

1. Registrul **contor de program** conține ..... a instrucțiunii care .....
2. Registrul instrucțiune o .....a instrucțiunii în .....
3. Partea de hardware conectează toate fanioanele done ale unităților la ..... prin efectuarea unei operații .....
4. CPU va fi atenționată despre ..... în intervalul de timp cât execută ....., urmând ca după terminarea acesteia să trateze ..... respectivă.
5. Manipulatorul întreruperilor testează ..... ale controllerelor de unitate pentru a determina care .....

M5.U1.7. I/O cu corespondenta in memorie (Memory-Mapped I/O)

Fiecare controller conține regiştri de control folosiți pentru a comunica cu procesorul; prin aceşti regiştri SO poate transmite comenzi sau poate consulta starea echipamentului. De asemenea, multe echipamente au o memorie tampon de date in care SO poate citi sau scrie

Exemplu. Memoria video este o memorie tampon de tip RAM, în care se scriu date ce se vor transforma în pixeli pe ecran.

Procesorul poate comunica cu regiştrii de control şi memoria tampon a unui echipament prin trei modalități, care vor fi prezentate în continuare.
(i) Fiecărui registru i se asociază un număr de PORT DE I/O, care este un întreg reprezentat pe `8` sau `16` biți; folosind instructiuni speciale de I/O ca: “IN REG, PORT” şi “OUT PORT, REG” procesorul poate copia registrul de control PORT în registrul său REG, respectiv poate scrie conținutul registruluin REG într-un registru de control.

Spațiul de adrese pentru memorie şi cel pentru I/O sunt diferite.

Exemplu. Instrucțiunile "IN REG,6" si "MOV REG,4" au semnificații diferite: prima citeşte în registrul REG conținutul portului 6; a doua citeste în REG cuvântul de memorie de la adresa 6; deci cei doi 6 se referă la zone de adrese de memorie diferite şi independente.

Aceasta metoda a fost folosita de cele mai multe dintre primele calculatoare, inclusiv IBM 360 si succesorii sai.

(ii) Regiştrii de control sunt puşi în spațiul de adrese de memorie. Fiecărui registru de control îi este repartizată o adresă unică de memorie, căreia nu-i corespunde o locație de memorie. Aceste sisteme se numesc I/O CU CORESPONDENTA IN MEMORIE (MEMORY-MAPPED I/O). Această metodă a fost introdusă de minicalculatorul PDP-11. De regulă adresele alocate regiştrilor de control se află la sfârşitul spațiului de adrese.

(iii) O schema hibridă foloseşte o memorie tampon de I/O în spațiul de adrese de memorie şi porturi de I/O pentru regiştrii de control;

Exemplu Pentium utilizează această arhitectură, cu adresele `640K – 1M` rezervate pentru memoriile tampon ale echipamentelor compatibile cu IMB PC şi porturi de I/O de la `0` la `64K`.

Toate aceste scheme functioneaza astfel:
- Când procesorul vrea să citească un cuvânt, fie din memorie fie dintr-un port de I/O, pune adresa dorită pe liniile de adresă ale magistralei (bus) şi apoi transmite un semnal de citire pe linia de control a magistralei. Pe o altă linie a magistralei pune specificația dacă este vorba de spațiul de adrese de memorie sau spațiul de I/O. Atunci fie memoria, fie echipamentul de I/O va raăpunde cererii.
- Daca exista doar spatiu de memorie (varianta (b)), atunci fiecare modul de memorie si fiecare echipament de I/O compara liniile de adresa cu domeniul de adrese pe care il foloseste; daca adresa se afla in domeniul sau, raspunde cererii (deoarece nici o adresa nu este repartizata atat memoriei cat si echipamentelor de I/O, nu exista ambiguitate).

Avantajele sistemului I/O cu corespondență în memorie. În cazul (i) sunt necesare instrucțiuni speciale de I/O pentru citirea/ scrierea la porturi (IN/OUT), care nu se regăsesc în limbajele C/C++, astfel încât este necesară folosirea limbajului de asamblare; în cazul (ii) regiştrii de control ai echipamentelor se pot asimila cu simple variabile alocate în memorie şi se pot accesa în C/C++ ca orice alta variabilă, deci driverele pentru echipamente pot fi scrise în aceste limbaje evoluate.

Nu este necesar un mecanism de protecție suplimentar pentru a împiedica procesele utilizator să efectueze operatii de I/O. E suficient ca SO să nu permită includerea porțiunii de spațiu de adrese ce conține regiştrii de control în spațiul virtual de adrese ale vreunui utilizator. Mai mult, dacă fiecare echipament are regiştrii de control în pagini diferite ale spațiului de adrese, SO poate permite unui utilizator să controleze anumite echipamente, incluzând paginile respective în tabela lui de pagini.În felul acesta, drivere diferite pot fi plasate îin spații de adrese diferite, reducându-se dimensiunea nucleului şi făcând totodată ca driverele să nu intefereze între ele.

Aceleaşi instrucțiuni care pot referenția memoria pot inițializa şi regiştrii de control.

Exemplu. Dacă există o instrucțiune "TEST" care testează dacă un cuvânt de memorie este 0, ea poate testa şi dacă un registru de control este 0, adică echipamentul este nefolosit şi poate accepta o nouă comandă. Codul în limbaj de asamblare ar putea arăta astfel:

```
LOOP: TEST PORT_4 // verifica daca portul 4 este liber
      BEQ READY // daca este 0, salt la eticheta
      BRANCH LOOP // altfel salt la LOOP (continua sa
testezi)
      READY:
```

Observație. În cazul (i) trebuie o instrucțiune în plus, care să citească registrul de control într-un registru al procesorului şi apoi sa fie testat, deci detectarea unui echipament nefolosit se va face mai lent.

Dezavantajele sistemului I/O cu corespondenta in memorie. Calculatoarele moderne au posibilitatea păstrării într-o memorie intermediară (cache) a unor cuvinte din memorie; dacă am păstra în memoria cache valoarea unui registru de control, atunci în codul assembler de mai sus prima referire la PORT_4 va face ca conținutul lui să fie pus în memoria cache, iar următoarele referiri vor lua valoarea din cache fara a consulta echipamentul (deci, cand echipamentul va fi la un moment dat pregatit, softul nu va afla ci va cicla la infinit); de aceea, în cazul (ii) hardware-ul trebuie să poată opri selectiv păstrarea în cache, cum ar fi pe baza numărului de pagină. Acest lucru adaugă complexitate suplimentară atat la hardware cât şi la SO, care va trebui să gestioneze păstrarea selectivă în cache.

Dacă există un singur spațiu de adrese, toate modulele de memorie şi toate echipamentele de I/O trebuie să examineze toate referințele la memorie pentru a vedea care din ele trebuie să raspundă; daca masina are o singura magistrala, acest lucru e simplu; daca are mai multe magistrale (o magistrala dedicata memoriei, care este mai rapida si nu ia în considerație faptul că echipamentele de I/O sunt mai lente), atunci echipamentele de I/O nu vad adresele ce trec prin magistrala memoriei, deci nu pot raspunde.

Soluția de rezolvare a problemei anterioare este de a avea mai multe magistrale, unele de mare viteză dedicate accesării memoriei interne, iar altele să fie dedicate operațiilor de I/O:

- referintele la memorie se transmit intai memoriei, apoi pe celelalte magistrale, pana cand cineva raspunde; este insa necesara o complexitate hardware suplimentara;

-se plaseaza pe magistrala memoriei un echipament de supraveghere ce transmite adresele catre echipamentele de I/O potential interesate; atunci insa s-ar putea ca echipamentele sa nu poata procesa cererile cu viteza cu care poate face acest lucru memoria;

- solutia folosita de Pentium, care are o magistrala locala, din care se ramifica 3 magistrale externe: memorie, PCI, ISA; pe magistrala locala exista un chip punte PCI (PCI bridge) la ramificatia catre magistrala PCI; chipul are niste registri de domeniu (range registers), preincarcati la pornirea sistemului (boot-are), prin care marcheaza anumite domenii de adrese (de ex. 640K - 1M) ca neapartinand memoriei; chipul filtreaza adresele venite pe magistrala locala, iar adresele din domeniile marcate sunt transmise magistralei PCI, nu memoriei; de asemenea, pe magistrala PCI exista un chip punte la ramificarea catre magistrala ISA (si mai lenta); dezavantajul este necesitatea de a descoperi la boot-are care sunt intervalele de adrese.

Accesul direct la memorie. În cazul accesării directe a perifericelor de către CPU cu testare periodică, CPU este utilizată pentru a transfera date între regiştrii de date ai controllerului şi memoria primară. Driverul de unitate copiază date din zona de date a procesului, în controller pentru fiecare operație de ieşire şi invers, pentru operațiile de intrare. În cazul utilizării întreruperilor, administratorul unității se ocupă cu sarcina de transfer. Ambele modalități presupun implicarea CPU, deci un consum al timpului său.

Multe dintre calculatoarele moderne folosesc un alt mecanism care cere o implicare a CPU mai redusă în efectuarea acestor operații, prin utilizarea unui procesor specializat numit **controller DMA** (**D**irect **M**emory **A**ccess). Mecanismul DMA presupune că operația de copiere în memorie să fie efectuată de către controller şi nu de către unitatea centrală.

Înțelegerea între controllerul DMA şi controllerul de unitate este realizată prin intermediul unei perechi de semnale numite cerere DMA şi confirmare DMA. Când are date de transmis, controllerul de unitate inițiază o cerere DMA şi aşteaptă până când primeşte o confirmare DMA. După ce s-a efectuat transferul, controllerul DMA semnalează terminarea operației printr-o întrerupere. Deşi controllerul DMA lucrează fără a ține ocupată CPU, totuşi cele două concurează pentru obținerea accesului la magistrală. Cu toate acestea, DMA creşte performanțele calculatorului, în sensul că operațiile de I/O se fac printr-o participare mai restrânsă a CPU.

Procesul de transfer al informațiilor de pe disc, folosind DMA se desfăşoară astfel:

1. Driverului de unitate i se comunică să transmită date în bufferul de la adresa `x`.
2. Driverul de unitate transmite controllerului de disc să transfere `c` cuvinte de memorie de pe disc în bufferul de la adresa `x`.
3. Controllerul de disc începe transferul DMA.
4. Controllerul de disc transmite fiecare cuvânt controllerului DMA.
5. Controllerul DMA transferă cuvintele primite în bufferul de la adresa `x`, incrementând adresa de memorie şi decrementând `c`, până când `c=0`.

6. Când `c=0`, controllerul DMA semnalează CPU prin intermediul unei întreruperi, că transferul de date s-a terminat.

Observații.
- Folosind DMA, controllerul de unitate nu trebuie să conțină obligatoriu regiştrii de date, deoarece controllerul DMA poate face transferul direct de la/la unitate.
- Controllerul trebuie să conțină un registru de adrese, în care se încarcă un pointer către un anumit bloc de memorie, de unde controllerul preia informații sau depune informații.
- De asemenea, trebuie să cunoască numărul de octeți ai blocului pe care trebuie să-l manevreze.
-Anumite calculatoare folosesc adrese de memorie fizică pentru DMA; altele, mai perfecționate execută transferul folosind adrese virtuale, adică adrese din memoria secundară, prin mecanismul denumit DVMA(Direct Virtual Memory Acces).

Să ne reamintim...

Fiecare controller conține regiştri de control folosiți pentru a comunica cu procesorul; prin aceşti regiştri SO poate transmite comenzi sau poate consulta starea echipamentului. De asemenea, multe echipamente au o memorie tampon de date in care SO poate citi sau scrie

Multe dintre calculatoarele moderne folosesc un alt mecanism care cere o implicare a CPU mai redusă în efectuarea acestor operații, prin utilizarea unui procesor specializat numit **controller DMA** (**D**irect **M**emory **A**ccess). Mecanismul DMA presupune că operația de copiere în memorie să fie efectuată de către controller şi nu de către unitatea centrală.

I. Înlocuiți zona punctată cu termenii corespunzători.

1. Fiecărui registru i se asociază ……., care este un ..... reprezentat pe .....
2. Fiecărui ..... îi este repartizată o adresă unică de memorie, căreia nu-i corespunde .....
3. Înțelegerea între ..... şi controllerul de unitate este realizată prin intermediul unei perechi de semnale numite ..... şi .....
4. Folosind DMA, ... nu trebuie să conțină obligatoriu ....., deoarece controllerul DMA poate face transferul .....
5. Controllerul trebuie să conțină un ....., în care se încarcă un ..... către un anumit ....., de unde controllerul preia ..... sau depune .....

M5.U1.8. Utilizarea zonelor tampon("buffering")

Utilizarea zonelor tampon la intrare ("input buffering") este tehnica prin care informațiile citite de la un dispozitiv de intrare sunt depuse într-o zonă de memorie primară, înainte ca procesul să le solicite pentru a le prelucra. Utilizarea zonelor tampon la ieşire ("output buffering") este metoda prin care datele de ieşire sunt salvate în memorie şi apoi scrise pe o anumită unitate, în acelaşi timp cu continuarea execuției procesului. Buffering-ul are ca scop lucrul în paralel al CPU şi unităților periferice.

Pentru a înțelege mai bine cum buffering-ul poate creşte performanțele sistemului de calcul, să considerăm câteva caracteristici ale proceselor. Anumite procese sunt orientate către operații de I/O, consumând o mare cantitate de timp pentru a efectua aceste operații. Alte procese sunt orientate spre calcule. Multe procese conțin faze în care sunt orientate către operații de I/O sau faze când sunt orientate către calcule.

Driverul poate gestiona unul sau mai multe buffere, unde depune caracterele citite, înainte ca acestea să fie prelucrate de către procesul care a inițiat cererea, respectiv de unde preia caracterele depuse de CPU(procesul utilizator) la scriere.

Numărul de buffere poate fi extins de la `2` la `n`.

Producătorul de date (controllerul în cazul operațiilor de citire, respectiv CPU pentru operațiile de scriere) scrie în bufferul `i` în timp ce consumatorul (controllerul în cazul operațiilor de scriere, respectiv CPU în cazul operațiilor de citire) citeşte din bufferul `j`.

În această configurație, bufferele de la `j` la `n-1` şi de la `0` la `i-1` sunt pline. În timp ce producătorul introduce date în bufferul `i`, consumatorul citeşte date din bufferele `j`, `j+1`,...,`n-1` şi `0,1,...,i-1`. Reciproc, producătorul poate umple bufferele `i`, `i+1,...,j-1` în timp ce consumatorul citeşte bufferul `j`.

În această tehnică de dispunere circulară a bufferelor, producătorul nu poate trece în zona bufferelor administrate de consumator, deoarece el poate scrie peste buffere care nu au fost încă citite. Producătorul poate scrie date numai în bufferele până la `j-1` în timp ce datele din bufferul `j` aşteaptă să fie prelucrate. În mod similar, consumatorul, nu poate trece în zonele administrate de producător deoarece el ar aştepta să citească informații, înainte ca acestea să fi fost plasate în zonele respective de către producător.

Observații. Numărul bufferelor de citire, respectiv de scriere trebuie ales în funcție de tipul proceselor. Pentru un proces orientat către I/O este mai potrivit să se aloce un număr cât mai mare de buffere de citire, pe când pentru un proces orientat spre calcule este necesar ca numărul de buffere de scriere să fie mai mare.

Orice proces poate fi la un moment dat orientat către I/O, ca mai târziu să devină orientat către calcule şi reciproc. Deci se impune un mecanism de comutare a procesului dintr-o stare în alta şi, corespunzător de modificare a configurației zonelor tampon.

Să ne reamintim...

Utilizarea zonelor tampon la intrare (“input buffering”) este tehnica prin care informațiile citite de la un dispozitiv de intrare sunt depuse într-o zonă de memorie primară, înainte ca procesul să le solicite pentru a le prelucra. Utilizarea zonelor tampon la ieşire (“output buffering”) este metoda prin care datele de ieşire sunt salvate în memorie şi apoi scrise pe o anumită unitate, în acelaşi timp cu continuarea execuției procesului. Buffering-ul are ca scop lucrul în paralel al CPU şi unităților periferice.

I. Înlocuiți zona punctată cu termenii corespunzători.

1. Driverul poate gestiona ....., unde depune ....., înainte ca acestea să fie prelucrate de către procesul care a inițiat cererea, respectiv de unde preia .....la scriere.

2. Numărul ..... de citire, respectiv de scriere trebuie ales în funcție de .....

3. Orice proces poate fi la un moment dat orientat către ....., ca mai târziu să devină orientat către ..... şi reciproc.

M5.U1.9. Test de evaluare a cunoştinţelor
Marcaţi varianta corectă.

1. Independența față de dispozitvele periferice a programelor înseamnă:

a)Acestea pot utiliza orice periferic..

b) Acestea nu trebuie să sufere modificări importante atunci când tipul dispozitivului periferic utilizat este altul decât cel prevăzut inițial.

c) Acestea nu trebuie să sufere modificări importante atunci când se adaugă un nou periferic.

d) Acestea nu trebuie să sufere modificări importante atunci când se elimină un periferic.

2. API furnizează:

a) Un set de apeluri către sistemul de operare.

b) Un set de apeluri către compilatoare.

c) Un set de funcții pe care un programator le poate apela pentru a utiliza o unitate.

d) Un set de funcții pe care un programator le poate apela pentru a utiliza un fişier.

3. Driverele sunt:

a) Componente hardware.

b) Componente software utilizate pentru execuția proceselor.

c) Componente ale SO folosite pentru gestiunea fişierelor.

d) Componente ale SO folosite pentru gestiunea perifericelor.

4. Operațiile de I/O se execută**:**

a) În mod utilizator.

b) În mod supervizor.

c) Prin lansarea în execuție a unui compilator.

d) Prin lansarea în execuție a unui link-editor.

5. În care dintre metodele de execuție a operațiilor de intrare/ieşire CPU este implicată cel mai mult:

a) Bazată pe testarea periodică a stării unității.

b) Bazată pe drivere şi întreruperi.

c) Bazată pe accesul direct la memorie.

d) Bazată pe salvarea regiştrilor UC.

6. Componenta software poate plasa o comandă în registrul de comenzi pentru a activa unitatea, atunci când fanioanele busy şi done sunt setate:

a) busy pe `0` şi done pe `1`.

b) Ambele pe `1`.

c) busy pe `1` şi done pe `0`.

d) Ambele pe `0`.

7. Care este componenta sistemului de operare de interfață cu regiştrii controlerului de unitate:

a) Driverul de unitate.

b) Întreruperea unității.

c) Compilatorul unității.

d) Link-editorul unității.

8. API este o interfață între:

a) Procesul(aplicația) în execuție şi drivere

b) Procesul(aplicația) în execuție şi CPU.

c) SO şi controller.

d) CPU şi unitatea de I/O.

9. Un driver se execută:

a) Ca parte a sistemului de operare. ☐

b) Prin lansarea în execuţie a unui compilator. ☐

c). Ca parte a programului. ☐

d) Prin lansarea în execuţie a unui link-editor. ☐

10. Interfaţa între controller şi unitate caută:

a) Să rezolve problema compatibilităţii între diverse unităţi. ☐

b) Să rezolve problema compatibilităţii între diverse sisteme de fişiere. ☐

c) Să rezolve problema compatibilităţii între unităţi fabricate de diverşi producători. ☐

d) Să optimizeze alocarea de spaţiu pe disc. ☐

M5.U1.10 Rezumat

Administrarea unei unități de I/O este implementată prin cooperarea dintre administratorului de resurse, driverelor de unități şi manipulatorului de întreruperi. În ceea ce priveşte modul cum se va face referire la o anumită unitate, proiectantul sistemului de calcul va alege între includerea acesteia în spațiul de adrese de memorie, sau alocarea unui spațiu separat, cu instrucțiuni specifice. Anumite sisteme permit accesul direct la memorie, astfel încât controllerul poate citi/scrie informații direct din/în memorie, fără intervenția procesorului.

Tendința actuală spre sisteme deschise, a încurajat proiectanții de calculatoare să creeze facilități prin care administratorii de sistem să poată adăuga noi unități în configurație sau drivere, fără a fi necesar să se modifice codurile sursă ale sistemelor de operare. Linux şi Windows sunt două exemple de sisteme de operare larg răspândite care care furnizează facilități de asistare a administratorului în configurarea de noi unități la adăugarea lor în configurația deja existentă.

Deoarece multe dintre fazele de execuție ale proceselor sunt orientate către efectuarea unor operații de I/O, perifericele fiind des solicitate de mai multe procese simultan, s-au depus eforturi considerabile de către proiectanții de sisteme de calcul pentru a creşte performanțele subsistemelor de intrare/ieşire.

Utilizarea zonelor tampon este o metodă tradițională folosită pentru a creşte performanțele sistemului, exprimate prin reducerea dimensiunii timpului de răspuns, prin suprapunerea efectuării operațiilor de I/O cu cele de calcul, atât la nivelul unui singur proces cât şi între procese.

Pornind de la necesitatea prelucrării unui volum mare de informații, unitățile de stocare a datelor sunt componente fundamentale ale calculatoarelor din zilele noastre, implicate din ce în ce mai mult în rezolvarea unor probleme de prelucrare. Deci tehnologiile hardware privind stocarea datelor se perfecționează din ce în ce mai rapid prin apariția de noi medii de stocare, a discurilor cu o înaltă densitate, cu un timp de acces rapid, ceea ce trebuie să conducă la îmbunătățiri ale componentelor sistemului de operare care le administrează.

Modulul 6. Studii de caz

Cuprins

Introducere.

În modulele anterioare am prezentat principiile, algoritmii, abstractizările şi în general conceptele fundamentale pe care se sprijină sistemele de operare. În acest modul, vom vedea cum sunt aplicate principiile prezentate în cazul a două dintre cele mai răspândite familii de sisteme de operare: Linux şi Windows.

Foarte multe persoane îşi pun întrebarea de ce este necesar studiul sistemelor Linux de către viitorii specialişti în informatică. Apărut la începutul anilor ’90, Linux este un continuator al sistemului Unix. Încă de la apariția sa, Unix a introdus concepte revoluționare în lumea sistemelor de operare şi a progresat odată cu evoluția sistemelor de calcul, bucurându-se de un succes deosebit în lumea specialiştilor. Datorită faptului că a fost creat de programatori pentru programatori, Unix a oferit facilități de operare mult superioare altor sisteme de operare. Datorită portabilității sale, el a fost utilizat pe cele mai diverse arhitecturi hardware. Din această cauză, a avut şi un real succes comercial.

Din punctul de vedere al utilizatorului, diferența esențială dintre Linux şi Unix, este că utilizarea Linuxului este liberă, pe când Unix este un produs comercial. Linux este unul dintre cele mai cunoscute produse „Open Source”. Aceasta este una dintre cauzele pentru care a progresat foarte rapid, s-a adaptat evoluțiilor din ştiința calculatoarelor şi a ajuns să implementeze marea majoritate a facilităşilor şi mecanismelor din Unix.

Sistemele de operare Windows este au fost create de compania Microsoft. Evoluția acestor sisteme, este legată de dezvoltarea calculatoarelor personale. Microsoft a introdus Windows pe piaţă pentru prima dată în noiembrie 1985, ca o interfață grafică pentru MS-DOS, iar la începutul anilor `90 a lansat prima versiune de sistem de operare Windows. Sistemele de operare Windows au cucerit o foarte mare parte a pieții. Se estimează că astăzi peste 85 % din calculatoarele Personale funcționează sub Windows. Windows oferă o interfață grafică prietenoasă şi mecanisme de gestiune a resurselor interne care îl fac competitiv. Întreaga filozofie a interfețelor Windows este subordonată conceptului de „birou electronic”.

Competențe

La sfârşitul acestui modul studenții vor fi capabili:

- să explice evoluția sistemelor Linux şi Windows;
- să explice interfețele cu utilizatorii a celor două sisteme de operare;
- să explice mecanismele interne ale Linux şi Windows, prin care sunt gestionate resursele sistemelor de calcul.

Unitatea de învățare M6.U1. Sistemele Linux

Cuprins

M6.U1.1. Introducere

Linux este un sistem multiuser si multitasking, adică mai mulți utilizatori pot rula mai multe programe în același timp. Oferă suport pentru lucrul în rețea, acces la Internet şi este unul dintre cele mai folosite sisteme de operare pentru administrarea serverelor din rețele de calculatoare.

Termenul Linux se referă şi la nucleul Linux, însă în mod uzual este folosit pentru a descrie întregul sistem de operare pentru calculatoare, compus din nucleul Linux, biblioteci software şi diverse unelte. O "distribuție Linux" adaugă acestor componente de bază o mare cantitate de programe, organizate în „pachete”. Nucleul Linux a fost dezvoltat inițial pentru microprocesorul Intel 386, dar în prezent rulează pe o mare gamă de microprocesoare şi arhitecturi de calculatoare. Este folosit pe calculatoare de tip personal, pe supercomputere, dar şi pe sisteme încapsulate, cum ar fi unele telefoane mobile sau recordere video.

Inițial dezvoltat şi utilizat de către programatori voluntari, Linux a câştigat suportul industriei IT şi al marilor companii ca IBM, Hewlett-Packard, Dell, Sun Microsystems, Google, Novell sau Nokia şi a depăşit ca folosire versiunile proprietare de Unix. Analiştii atribuie succesul sistemului faptului că este independent de furnizor, implementarea are un cost scăzut iar securitatea şi fiabilitatea sistemului sunt considerate de către specialişti drept foarte bune.

M6.U1.2. Obiectivele unității de învățare

Această unitate de învățare îşi propune ca obiectiv principal o inițiere a studenților în conceptele generale asupra sistemelor Linux. La sfârşitul acestei unități de învățare studenții vor fi capabili să:

- înțeleagă şi să explice evoluția sistemelor Linux;
- înțeleagă şi să explice organizarea sistemelor Linux;
- înțeleagă şi să explice structura nucleului Linux;
- înțeleagă şi să explice interfețele oferite de sistemul Linux;
- înțeleagă şi să explice organizarea sistemului de fişiere sub Linux;
- înțeleagă şi să explice administarea proceselor sub Linux;
- înțeleagă şi să explice gestiunea memoriei sub Linux.

Durata medie de parcurgere a unității de învățare este de 3 ore.

M6.U1.3. Concepte introductive.

Evoluția sistemelor de operare Unix şi Linux. În 1969 apare prima versiune a sistemului de operare UNIX. Această realizare este legată de două evenimente importante:
- realizarea de către MIT (**M**assachusets **I**nstitute of **T**echnology) a produsului MULTICS, care a implementat o serie de indei noi, printre care: introducerea unui interpretor de comenzi care să fie un proces utilizator; fiecare comandă să fie un proces; introducere unor caractere de interpretare a liniilor; sistem multi-utilizator şi interactiv.
- în cadrul firmei Bell Labs (laboratoarele de cercetare ale firmei AT&T) a fost lansat un proiect de cercetare care a avut ca efect realizarea unui **sistem ierarhic** de organizare a fişierelor.

Aceste concepte au stat la baza primei versiuni a sistemului UNIX, scris în limbaj de asamblare şi implementat pe un sistem PDP-7.

În 1971 apare următoarea versiune a sistemului de operare UNIX. Acesta a fost implementat pe un calculator mai puternic (PDP-11/20). Este rescris pentru noul sistem şi apare o documenrație a sistemului. În 1974 sunt prezentate în *Communications of the ACM* ideile revoluționare ale acestui nou sistem de operare. De asemenea, au fost dezvoltate încă trei versiuni ale sistemului şi autorii sistemului îl pun gratuit la dispoziția universităților americane şi au fost furnizate licențe de UNIX unor instituții comerciale.

În 1976 apare versiunea 6, care a fost prima lansată în afara Bell Labs şi ale cărei componente sunt majoritar scrise în limbajul C. În felul acesta, sistemul devine **portabil**, adică poate fi instalat cu eforturi minime pe platforme de calcul diferite.

După această dată, sunt dezvoltate în paralel, de către Universitatea Berkeley şi Bel Labs diverse versiuni, în concordanță cu evoluția sistemelor de calcul. Versiunea 7, apărută în 1978 a devenit comercială, iar din punct de vedere conceptual a adăugat organizarea sistemului de I/O sub formă de fluxuri, care permite o configurare mai flexibilă a administrării comunicației între procese, precum şi un sistem de fişiere care să permită apelul de proceduri la distanță. Ea a fost predecesorul UNIX BSD (Berkeley Software Distribution). Dintre următoarele implementări, cele mai semnificative sunt BSD 4.1, BSD 4.2, BSD 4.3 realizate la Universitatea Berkeley şi Sistem III, Sistem V ale Bell Labs.

Următoarele relizări au încercat să standardizeze sistemul, ceea ce adus la o formă general acceptată denumită UNIX System V Release 4 (prescurtat SVR4). Şi alte firme realizează propriile versiun de UNIX, care respectă standardele SVR4, dintre care cele mai semnificative sunt Ultrix şi OSF(firma DEC- Digital Equiment Corporation), HP/UX(firma Hewlett-Packard), IRIX(firma SGI-Silicon Graphics Incorporated), AIX(firma IBM), SunOS şi Solaris(firma SUN Microsystems) etc. Dintre facilitățile adăugate de aceste versiuni se remarcă introducerea interfețelor grafice cu utilizatorul şi cele legate de lucrul în rețele de calculatoare.

Standardizarea diferitelor variante de UNIX s-a realizat sub auspiciile unui comitet de standarde al IEEE. Acest proiect a fost denumit POSIX (**P**ortable **O**perating **S**ystem UN**IX**) şi, în timp a devenit o colecție de standarde pe care le respectă toate sistemele de operare moderne. El este format din mai multe componente, dintre care cele mai importante sunt:

- POSIX.1 – defineşte procedurile care cer apeluri de sistem;
- POSIX.2 – se referă la interfața cu sistemul(shell);
- POSIX.3 – metode de testare a sistemului;
- POSIX.4 – componentele specifice ale sistemelor de operare pentru calculatoare în timp real;
- POSIX.6 – elemente de securitate;

- POSIX.7 – elemente de administrare

etc.

În 1987 a apărut sistemul MINIX, lansat în scopuri didactice la Universitatea Vrije din Amsterdam, care este compatibil cu UNIX la nivel de comenzi şi apeluri de sistem, dar este original din punctual de vedere al proiectării şi implementării.

În 1991, pornind de la MINIX, apare prima versiune (0.01) a sistemului LINUX, realizat de un student finlandez, Linus Torvalds. El a fost creat pentru sistemele de calcul pe 32 de biți, compatibile IBM PC. Sursele lui au fost puse la dispoziția tuturor celor interesați, fiind libere pe Internet. Fiind un produs Open source, s-a dezvoltat rapid ajungând o adevărată clonă a sistemului Unix. Marea majoritate a funcționalităților Unix au fost implementate pe Linux.
Următoarea versiune importantă(1.0) a fost lansată în 1994; a venit cu un nou sistem de fişiere (inclusiv maparea fişierelor în memorie), a introdus suportul pentru lucru în rețea (inclusiv Internet), a inclus noi drivere de dispozitiv. O altă versiune importantă(2.0) a fost lansată în 1996; a inclus suportul necesar arhitecturilor pe 64 de biți, multiprogramare simetrică, noi protocoale pentru lucru în rețea etc. Foarte multe componente software au fost portate din Unix, printre care şi aplicația de interfață grafică X Window System.

Evoluția sistemului LINUX a fost încurajată de majoritatea firmelor de software importante (cu excepția Microsoft), o parte dintre ele oferind-ul la prețuri foarte mici. Urmând filozofia UNIX-ului, LINUX este mai degrabă un nucleu, care implementează funcțiile de bază, fiind deschis tuturor celor care doresc să adauge noi module care să extindă sistemul.

Caracteristici generale ale sistemelor Linux

- **Linux**-ul este gratuit, spre deosebire față de Unix.
- **Linux**-ul este un sistem de operare multi-user şi multi-tasking ce oferă utilizatorilor numeroase utilitare interactive.
- **Linux** este un mediu de dezvoltare; când ne referim la **Linux,** ne referim la nucleu (kernel), care are rolul de a gestiona memoria şi operatiile I/O de nivel scazut, precum şi planificarea şi controlul execuției diferitelor task-uri (procese). Utilizatorii au la dispozitie un număr mare de utilitare pentru munca lor: editoare de text, limbaje de comandă (shell-uri), compilatoare, debugger-e, sisteme de prelucrare a textelor, protocoale diverse pentru acces la Internet etc.
- UNIX-ul a apărut în perioada când sistemele de operare foloseau prelucrarea serială a lucrărilor. De la început el a fost un sistem interactiv, cu divizarea timpului. **Interactivitatea** sistemului este facilitată şi de interfața cu utilizatorul, care poate fi înlocuită sau dezvoltată, fiind ea însăşi un proces. Bazându-se pe divizarea timpului, UNIX a fost primul sistem de operare **multi-utilizator şi multi-proces**.
- Linux este un sistem **portabil**; perifericele sunt tratate ca fişiere iar driverele sunt singurele componente ale sistemului care sunt scrise in limbaj cod maşină.

Organizarea ierarhică a sistemelor Unix/Linux. În figura 6.1.1 este prezentată organizarea pe niveluri a unui sistem care lucrează sub Unix sau Linux. Sistemul oferă o interfață de comunicare atât cu utilizatorii, cât şi cu diverse aplicații ale acestora.

Nucleul Linux. Orice proces lansat în execuție în mod utilizator interacționează cu nucleul prin intermediul apelurilor de sistem, care pot fi grupate în mai multe categorii:

- apeluri de sistem pentru accesarea sistemului de fişiere ;
- apeluri de sistem pentru lucru cu procese ;
- apeluri de sistem legate de planificarea execuției proceselor ;
- apeluri de sistem pentru comunicarea între procese;
- alte diverse apeluri.

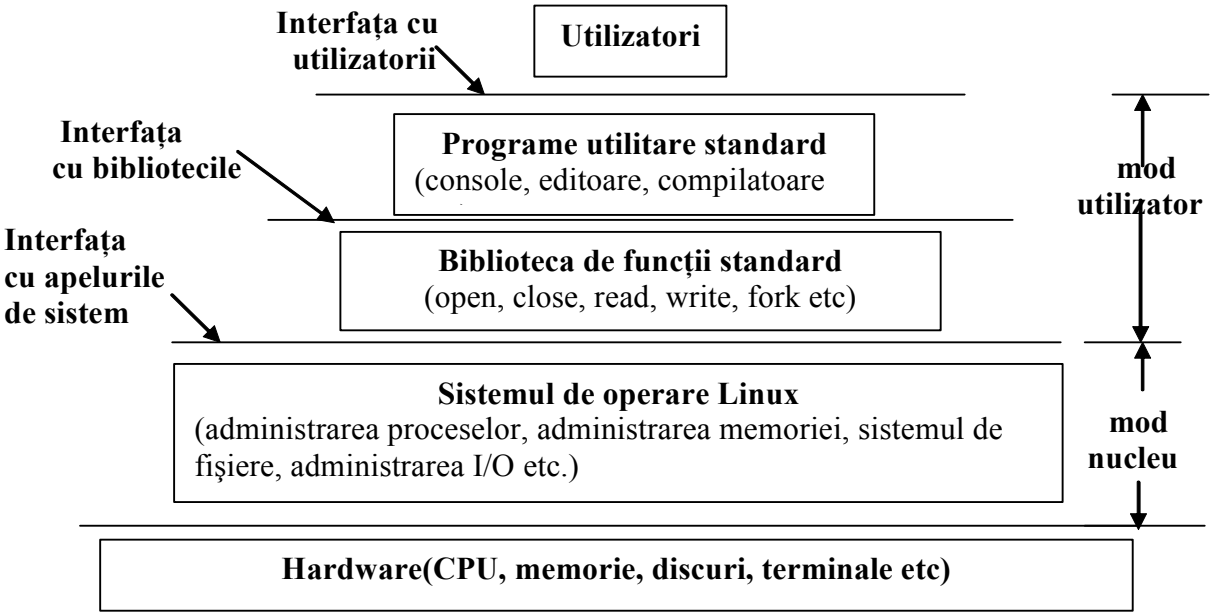


Figura 6.1.1 Organizarea sistemului sub Linux

Din clasificarea prezentată, rezultă că nucleul realizează următoarele funcții:

- Gestiunea proceselor şi a memoriei: prin apeluri de sistem, un proces cere serviciile componentei de creare, administrare şi planificare a execuției proceselor; aceasta intră în contact cu componenta de administrare şi alocare a memoriei virtuale, care lucrează împreună cu componenta de administrare a paginilor din memoria fizică, realizându-se astfel încărcarea paginilor virtuale în pagini fizice; de asemenea, prin intermediul CPU se transmite un semnal procesului cu privire la rezolvarea apelului solicitat.
- Accesarea controlată a dispozitivelor de tip caracter şi bloc prin intermediul driverelor.
- Gestiunea fişierelor: prin apeluri de sistem, un proces cere serviciile sistemului de fişiere, care gestionează o structură ierarhică de nume de directoare şi fişiere, cu informații despre fiecare entitate a structurii respective; pe baza acestor informații, folosind driverele de dispozitiv se accesează un anumit periferic, pe care se află fişierul căutat, unde se execută operația cerută în apelul de sistem specificat.
- Accesarea mediului de rețea prin intermediul protocoalelor de rețea, care accesează driverele dispoziticvelor specifice lucrului în rețea.

Distribuții Linux. Sistemele de operare Linux sunt disponibile sub formă de "distribuții". Unele dintre acestea sunt orientate spre utilizatorul particular, altele către servere sau către utilizatorii cu calculatoare mai vechi. Cele mai folosite distribuții de Linux sunt:
- Ubuntu este un proiect orientat spre utilizatorul, uşor de utilizat şi configurat, fiind în acelaşi timp puternic şi stabil. O distribuție înrudită cu ubuntu este Kubuntu, care foloseşte KDE.
- **SuSE Linux** este o distribuție orientată atât către servere cât şi spre stații de lucru sau desktopuri, fiind caracterizată de uşurința în utilizare şi configurare.
- **Fedora** este continuatoare a proiectului **Red Hat Linux**, care conține software liber şi disponibil gratuit de pe **Internet.**
- **Debian GNU/Linux** este una din distribuțiile cele mai complexe care conține un număr foarte mare de pachete de programe, precum şi un manager propriu de pachete.
- **Mandriva Linux** (denumită anterior **Mandrake Linux**) este o distribuție uşor de utilizat, orientată spre utilizatorii desktop; a fost creată de compania franceză **Mandriva.**

- **PCLinuxOS** este o distribuție derivată din **Mandriva Linux**, destinată mediului desktop şi care se remarcă prin usurința instalării, fiind adecvată pentru utilizatorii începători.
- **Slackware Linux** este una din cele mai vechi distribuții. Distribuției îi lipsesc unelte de configurare uşoară, dar beneficiază de viteză mare de rulare, posibilitate a de a fi instalată pe hardware mai vechi şi o organizare simplă a sistemului.
- **Gentoo Linux** este destinată utilizatorilor avansați, fiind orientată spre realizarea şi rularea unor aplicații complexe.
- **Knoppix** este o distribuție „live" care rulează direct de pe **CD** sau **DVD**, fără a instala nimic pe hard disk, ce poate fi utilizată, pentru diagnosticări de sistem, reparări, recuperări de date etc.
- **RedHat Linux** este una din cele mai cunoscute distribuții; în prezent este o distribuție comercială, orientată exclusiv spre piața serverelor şi spre mediul de afaceri. Este distribuția care a dat naştere proiectului **Fedora Core.**
- **Slax** este o distribuție "live" bazată pe **Slackware**, care poate rula de pe CD sau DVD sau de pe o memorie Flash de 256 MB.

Să ne reamintim...

În 1969 apare prima versiune a sistemului de operare UNIX. La baza lui a stat produsul MULTICS, care a introdus un interpretor de comenzi care să fie un proces utilizator şi fiecare comandă să fie un proces şi avut ca efect realizarea un sistem ierarhic de organizare a fişierelor.

LINUX este un continuator al UNIX, dar spre deosebire de acesta ete un produs liber, fiind puse la dispoziția tuturor celor interesați pe Internet. Multe dintre funcționalitățile Unix au fost portate pe Linux.

Linux -ul este un sistem de operare multi-user şi multi-tasking ce oferă utilizatorilor numeroase utilitare interactive.

Unix şi Linux au fost proiectate de programatori pentru programatori. Pornind de la acest fapt, **interactivitatea** şi **facilitățile de dezvoltare ale programelor** au fost elemente prioritare ale proiectanților sistemului. **Interactivitatea** sistemului este facilitată şi de interfața cu utilizatorul, care poate fi înlocuită sau dezvoltată, fiind ea însăşi un proces.

Sistemele de operare Unix/Linux au organizarea ierarhică, pe niveluri. Sistemul oferă o interfață de comunicare atât cu utilizatorii, cât şi cu diverse aplicații ale acestora.

Nucleul Linux este monolitic, adică, pentru a rezolva anumite necesități ale sistemului, el se execută ca unic proces, într-un singur spațiu de memorie.
Orice modificare a nucleului, presupune o nouă editare de legături, reinstalarea şi, evident o nouă încărcare ale sistemului de operare.

Sistemele de operare Linux sunt disponibile sub formă de distribuții. Cele mai folosite distribuții de Linux sunt: **Ubuntu, SuSE Linux, Fedora, Debian GNU/Linux, Mandriva Linux, PCLinuxOS, Slackware Linux, Gentoo Linux, Knoppix, RedHat Linux, Slax, NimbleX, TFM/GNU Linux**

I. Întrebări.

1. Care sunt cele mai importante facilități introduse de versiunile Linux care respectă standardele SVR4.
2. Care versiune de Linux a introdus suportul pentru lucru în rețea.
3. Ce versiune de Linux a portat din Unix interfața grafică X Window System.
4. Prin ce se asigură portabilitatea sistemului Linux.

5 La ce componente ale unui sistem Linux au acces utilizatorii.
6.Ce operații trebuie să se efectueze atunci când se face o modificare a nucleului.
7. Cum se face legarea la nucleu a unor module noi.
8. Cum interacționează procesele cu nucleul.
9. Explicați organizarea ierarhică a sistemelor Linux.
10. Care este considerată una dintre cele mai populare distribuții de Linux.
11. Specificați distribuțiile care rulează direct de pe CD sau DVD.
12. Care distribuție este continuatoarea liberă a proiectului RedHat.

M6.U1.4. Interfața sistemului cu utilizatorii

Interfața cu utilizatorii se poate face în mod text sau grafic. Componenta care permite acest lucru este **consola**, care joacă un rol esențial în introducerea şi afişarea de text folosit în utilizarea şi administrarea sistemului. Terminalele virtuale sunt utilizate pentru introducerea de comenzi sub formă de texte.

Interfețe grafice. Interfața grafică oferită de sistemele Linux se bazează pe **X Window System** (se utilizează şi denumirea scurtă **X** sau **X11**); acesta este un cadru ce permite desenarea de ferestre pe un afişaj în care imaginea este reprezentată ca o matrice de puncte.

X oferă un cadru de bază pentru crearea unor interfețe grafice: desenarea şi mutarea ferestrelor pe ecran, interacțiunea cu mouse-ul, interacțiunea cu tastatura. Restul de primitive sunt conținute în diverse programe de interfață, aceasta fiind cauza pentru care aspectul vizual diferă de la o distribuție Linux la alta sau chiar în cadrul aceleiaşi distribuții.

X Window System are o arhitectură de tip client-server. Serverul X primeşte de la clienți (procese) cereri de desenare a unor obiecte grafice şi trimite înapoi la aceştia intrările de la utilizator, preluate de la tastatură, mouse etc. În acest sens, serverul X este responsabil de administrarea resurselor care permit interacțiunea cu utilizatorul (ecranul, tastatura, mouse-ul etc.). Serverul X rulează pe calculatorul la care s-a conectat utilizatorul; el preia date de intrare (cereri) ale acestora, pe care le trimite aplicațiilor client, care procesează mesajele primite de la serverul X şi îi returnează acestuia rezultatul prelucrării, pe care acesta, mai departe îl transmite către utilizator prin afişare pe ecran. Clienții X pot rul local, pe acelaşi sistem cu serverul X, la care s-a conectat utilizatorul sau pe un alt calculator din rețea.

Față de aplcațiile clasice de tip client-server, aici avem o diferență, în sensul că clienții se pot afla pe calculatorul local sau la distanță, iar serverul se execută pe calculatorul local. Serverul X nu conține nici o specificație legată de felul cum arată interfața cu utilizatorii (modul în care arată butoanele, ferestrele, meniurile etc.); aceste specificații sunt realizate de către alte componente: **administratorul de ferestre** (window manager), **mediul de ecran** (desktop environment) şi **aplicațiile specifice de interfață grafică cu utilizatorul** (GUI specific application).

Administratorul de ferestre controlează amplasarea şi modul în care arată ferestrele aplicațiilor; astfel, toate ferestrele clienților au caracteristici comune: bara de titluri a ferestrei, butoanele de maximizare şi minimizare etc. Aspectul vizual al ferestrelor este controlată de o componentă a administratorul de ferestre, numită **window decorator**.

Exemplu.
Kwin este managerul de ferestre folosit de KDE; **metacity** este managerul de ferestre folosit de gnome. Despre gnome şi KDE vom discuta în cele ce urmează.

Mediul de ecran include un administrator de ferestre, mai multe aplicații şi o interfață vizuală atractivă. Marea majoritate a sistemelor Linux, pun la dispoziția utilizatorului două medii de ecran: **Gnome** şi **KDE**.

Interfaţa GNOME constă dintr-un **panel** şi un **desktop.** Panelul se găseşte în partea de sus a ecranului şi conţine **meniuri**, **progame** şi **appleturi.** Un ecran GNOME afişează trei meniuri: **Applications**, **Places** şi **System**.

Din meniul **Applications** se poate selecta orice orice aplicaţie care se execută pe calculatorul respectiv. Cu meniul **Places** se pot accesa câteva locaţii foarte utilizate: directorul gazdă al utilizatorului (despre care vom discuta în al doilea modul), catalogul **desktop**, care conţine fişierele afişate pe ecran şi fereastra **Computer** prin care putem accesa dispozitivele periferice, fişierele partajate din sistem şi toate directoarele din calculatorul respectiv. Meniul **System** include sub-meniurile **Preferences** şi **Administration**. Meniul **Preferences** este folosit pentru configurarea GNOME.

Pentru a muta o fereastră, cu click stânga pe mouse se trage fereastra respectivă. Fiecare fereastră are butoane pentru **maximizare**, **minimizare** şi **închidere**. Cu un Dublu-click pe bara de titluri maximizăm fereastra. Fiecare fereastră are un buton corespunzător în panelul de subsol. Putem folosi acest buton pentru a minimiza sau restaura fereastra respectivă. Interfaţa GNOME suportă facilităţile "drag-and-drop"; se pot trage pe ecran sau dintr-o fereastră în alta: directoare, icon-uri şi aplicaţii .

GNOME furnizează tool-uri de configurare. Acestea sunt afişate în meniul **System → Preferences,** fiind organizate în mai multe sub-meniuri: **Personal**, **Look and Feel**, **Internet and Network**, **Hardware** şi **System**. Cele ce nu se încadrează în niciuna dintre categoriile enunţate, sunt afişate direct. Fiecare asfel de fereastră are un buton **Help,** care prezintă o descriere a acesteia. Unele dintre cele mai importante astfel de tool-uri sunt:
- Personal → Keyboard Shortcuts realizează corespondenţe între taste şi anumite task-uri.
- Personal →File Management determină modul de afişarea al directoarelor şi fişierelor.
- Look and Feel → Windows permite adăugarea de facilităţi sporite în manipularea ferestrelor (definirea unor taste pentru modificarea ferestrelor, selectarea cu mouse-ul a ferestrelor etc.)

Utilizarea applet-urilor **GNOME.** Pentru adăugarea applet-urilor, la panelul respectiv se dă click dreapta pe acesta şi se selectează intrarea **Add**, care listează toate applet-urile disponibile. Câteva dintre cele mai folositoare applet-uri sunt: căutare în dicţionar, gradul de utilizare a CPU, verificarea funcţionării monitorului etc.

Interfaţa grafică KDE afiăează un panel în partea de jos a ecranului care are un aspect similar cu partea de sus a ecranului în cazul GNOME. **Administratorul de fişiere** apare puţin diferit, dar operează în acelaşi mod cu cel din GNOME. În meniul principal, există o componentă **Control Center**, din care se pot configura fiecare componentă KDE. Într-un mod asemănător cu GNOME, se pot selecta applet-uri, pentu utilizări identice.

Interpretorul de comenzi. Sub Linux, atât comenzile cât şi interpretoarele de comenzi (shell) sunt procese care se execută în spaţiul de memorie utilizator. Există mai multe interpretoare de comenzi care funcţionează sub Linux; cele mai utilizate sunt `sh`, `tcsh`, `bash`.Pe un sistem de calcul, se poate lansa în execuţie oricare dintre shell-urile amintite. După lansarea în execuţie, shell-ul deschide un terminal virtual, de la care utilizatorul poate introduce propriile comenzi, recunoscute de limbajul de operare al sistemului. Aceste comenzi devin procese, care sunt procese fiu ale shell-ului. Despre noţiunea de proces fiu vom discuta în modulul patru.

Pentru a se deschide un terminal virtual, se selectează din interfaţa grafică **Applications -> System Tools -> Terminal.** Implicit, acesta va afişa o linie care conţine numele de user, introdus când utilizatorul s-a conectat la sistem, numele directorului gazdă (despre care vom discuta în modulul următor) şi prompterul (de obicei caracterul `$`, pentru utilizatorul

obişnuit, respectiv #, pentru utilizatorul cu drepturi de root) care indică locul de unde se poate introduce o comandă. Caracterul utilizat ca prompter şi mesajul care apare înaintea lui, pot fi setate de utilizator, prin actualizarea unei variabile de sistem, care va fi descrisă în modulul trei.

Forma unei comenzi Linux. În intrducerea şi editarea comenzilor se pot utiliza anumite combinaţii de taste, prezentate în tabelul 6.1.1.

Tasta comb. de taste	**Efectul**
Ctrl-A	Mută cursorul la începutul liniei de comandă
Ctrl-C	Termină executia programului şi afişează prompterul
Ctrl-D	Se iese din sesiunea shell curentă (acelaşi efect cu comenzile **exit** sau **logout)**
Ctrl-E	Mută cursorul la sfârşitul liniei de comandă
Ctrl-H	Generează caracterul **Backspace**
Ctrl-L	Şterge terminalul curent
Ctrl-R	Caută istoricul comenzilor introduse la acel terminal
Ctrl-Z	Suspendă execuţia unui program
Săgeată stânga / Săgeată dreapta	Mută cursorul pe linia de comandă cu o poziţie la stânga, respectiv dreapta, loc în care se pot insera caractere
Săgeată sus/ Săgeată jos	Afişează comanda anterioară sau următoarea
Tab	Terminarea comenzii sau a numelui de fişier
Tab Tab	Arată fişierul sau posibilităţile de terminare a comenzii

Tabelul 6.1.1. Combinaţii de taste

O comanda Linux are 4 zone sau câmpuri:

- Primul câmp conţine **numele** propriu-zis al comenzii, care indică ce acţiune realizează comanda respectivă.
- Al doilea este **câmpul de opţiuni**; o opţiune este de obicei o literă, care poate fi urmată de un argument şir de caractere sau un număr întreg. Un grup de opţiuni poate începe cu semnul `- sau +.`
- **Zona de expresii conţine un** şir de caractere cerute ca argumente pentru comanda respectivă.
- **Zona de fişiere** conţine numele fişierelor utilizate în cadrul comenzii lansate.

Obţinerea de informaţii despre comenzi. Utilizarea paginilor manualelor se face cu comanda:

```
man [optiuni]nume_comanda
```

în care `nume_comanda` este numele comenzii despre care dorim să aflăm informaţii. Informaţiile obţinute sunt complexe, dar bine structurate. Dacă se tastează comanda `man man` se obţine o imagine a modului cum sunt structurate informaţiile din manualele Linux. Afişarea paginilor din manuale, relativ la comanda specificată ca argument, se face la terminalul de la care a fost tastată comanda `man`.

Exemplu. Comanda `passwd` este utilizată pentru schimbarea parolei de către un utilizator. Informaţii despre această comandă putem găsi în secţiunile 1 şi 5. Dacă tastăm: `man 1 passwd`

Sunt afişate informaţiile din secţiunea 1, cea cu numărul cel mai mic. Acelaşi efect se obţine dacă se tasteză `man 1 passwd`.

Cu `man -a passwd`, se afişează poate informaţiile despre comanda `passwd`.

Utilizarea comenzii Info. Informațiile obținute cu această comandă sunt mai recente şi mai uşor de utilizat În multe cazuri, paginile din manuale fac trimitere la paginile info. Prin tastarea comenzii `info info` sunt prezentate informații despre comanda info .

Cu comanda **`whatis`** se pot obține explicații scurte despre comenzi. De asemenea, este afişată secțiunea de manual în care se găsesc informații despre comanda respectivă. Cu opțiunea **`-help,`** care urmează o comandă, se obțin explicații scurte despre utilizarea comenzii şi o listă a opțiunilor disponibile.

Să ne reamintim...
Interfața cu utilizatorii se poate face în mod text sau grafic. Componenta care permite acest lucru este **consola**. Terminalele virtuale sunt utilizate pentru introducerea de comenzi sub formă de texte.Interfața grafică oferită de sistemele Linux se bazează pe **X Window System**; acesta este un cadru ce permite desenarea de ferestre pe un afişaj în care imaginea este reprezentată ca o matrice de puncte.

Mediul de ecran include un administrator de ferestre, mai multe aplicații şi o interfață vizuală atractivă. Cele mai utilizate sunt **Gnome** şi **KDE**. Sub Linux, atât comenzile cât şi interpretoarele de comenzi (shell) sunt procese care se execută în spațiul de memorie utilizator.

Utilizarea paginilor manualelor se face cu comanda `man nume_comanda`, în care `nume_comanda` este numele comenzii despre care dorim să aflăm informații. Afişarea paginilor din manuale, relativ la comanda specificată ca argument, se face la terminalul, de la care a fost tastată comanda `man`. Aceste pagini sunt organizate pe secțiuni. Comenzile pot avea pagini în mai multe secțiuni (paginile din secțiunea utilizatori, paginile din secțiunea administratorului de sistem, paginile din secțiunea programatorilor).

Informațiile obținute cu comanda `info` sunt mai recente şi mai uşor de utilizat În multe cazuri, paginile din manuale fac trimitere la paginile info.

Cu comanda `whatis` se pot obține explicații scurte despre comenzi. Cu opțiunea `–help,` care urmează o comandă, se obțin explicații scurte despre utilizarea comenzii şi o listă a opțiunilor disponibile.

I. Întrebări.
1. Care este deosebirea dintre serverul X şi o aplicație client-server obişnuită.
2. Care componente conțin specificațiile legate de felul cum arată interfața cu utilizatorii.
3. Cum se lansează un terminal virtual.

M6.U1.5. Tipuri de fişiere şi sisteme de fişiere

În cadrul unui sistem de fişiere, apelurile sistem Linux gestionează următoarele tipuri principale de fişiere:
- *Fişierele obişnuite* sunt privite ca şiruri de octeți; accesul la un octet se face fie secvențial, fie direct prin numărul de ordine al octetului.
- *Fişierele directori*. Un fişier director se deosebeşte de un fişier obişnuit numai prin informația conținută în el. Un director conține lista de nume şi adrese pentru fişierele subordonate lui. Uzual, fiecare utilizator are un director propriu care punctează la fişierele lui obişnuite, sau la alți subdirectori definiți de el.

- *Fişierele speciale*. Linux priveşte fiecare dispozitiv de I/O ca un fişier de tip special. Aceste fişiere sunt de tip bloc sau caracter. Din punct de vedere al utilizatorului, nu există nici o deosebire între lucrul cu un fişier disc normal şi lucrul cu un fişier special. Se realizează astfel simplitatea şi eleganța comenzilor şi numele unui dispozitiv poate fi transmis ca argument.
- *Fişierele legături simbolice* sunt un fel de alias-uri pentru fişiere.
- *Fişierele fifo* sunt folosite pentru comunicația între procese care se execută pe acelaşi calculator.
- *Fişierele socket* sunt folosite pentru comunicația între procese care se execută într-o rețea de calculatoare.

Tipuri de sisteme de fişiere. Sistemele de fişiere cele mai întâlnite sunt de tipul *ext*; *ext3* este continuatorul lui *ext2*, care la rândul lui este o perfecționare a lui *ext*. Sistemul *ext* a fost unul din primele sisteme fisiere din Linux; el recunoaşte sistemele de fişiere din UNIX. El a a fost înlocuit de *ext2*. Acesta a fost unul dintre cele mai rapide sisteme de fişiere. El a introdus suportul pentru volume pâna la 4 TB, gestiunea numelor lungi si suportul complet al fisierelor UNIX. Pentru a putea fi compatibil cu viitoarele versiuni, utilizeaza **hook**, un fel de plug-in ce permite extinderea functionalităților, mentinând structura de bază a sistemelor de fişiere. *ext3* introduce caracteristici de journalizare pentru metadate (atributele fişierului) sau pentru copierea fişierului în timpul fazei inițiale de actualizare a jurnalului; această ultimă opțiune, garantează o mai bună recuperare a datelor.

Directori şi inoduri. O intrare într-un fişier director conține *numele fişierului* şi *numărul nodului de index* (inod) al acestuia. Inodul (nodul de index) reprezintă o structură de date care conține informațiile despre un anumit fişier (sub-director) din directorul respectiv. Informațiile cele mai importante conținute în nodul de index sunt sunt:
- Drepturile de acces şi tipul fişierului.
- Numărul de legături spre acest fişier (contor de legare).
- Numărul (UID) de identificare al proprietarului(user ID).
- Numărul (GID) de identificare a grupului(group ID).
- Lungimea fişierului.
- Momentul ultimului acces la fişier.
- Momentul ultimei modificări a fişierului.
- Momentul ultimei modificări a structurii inodului.
- Lista adreselor disc pentru primele blocuri care aparțin fişierului.
- Referințe către celelalte blocuri care aparțin fişierului.

Structura arborescentă de directoare şi fişiere. Directoarele se deosebesc de un fişier obişnuit prin informațiile conținute. Fiecare disc conține un director propriu numit **rădăcină** (Root), specificat prin \. Orice director conține lista de nume şi adrese pentru fişierele subordonate lui. Se crează astfel o structură arborescentă a sistemului de fişiere corespunzător unui disc.

Orice utilizator are un **director gazdă (home directory)** în care îşi poate crea propria structură de directoare şi fişiere, ataşat utilizatorului la intrarea în sistem, la care se poate face referință prin caracterul ~. Directorul are două intrări speciale:
- "." conține o referință către însuşi directorul respectiv;
- " .. " conține un pointer către directorul părinte.

La intrarea în sistem**, directorul curent** este directorul gazdă**.** Acesta se poate modifica şi referința la el se face prin caracterul "." .

În cadrul unui sistem de fişere, orice fişier(sau director), este identificat prin calea către el. Calea reprezintă un drum în structura arborescentă de fişiere a discului respectiv. O cale poate

fi absolută (dacă porneşte din directorul rădăcină) sau relativă (dacă porneşte dintr-un alt director). **Specificarea unei căi** este de forma:

```
[\]dir_1\dir_2\...\dir_n\nume
```

Dacă directorul rădăcină lipseşte, atunci calea este relativă. În plus faţă de MS-DOS sau Windows, este permisă şi structura de **graf aciclic**. Astfel, un fişier poate aparţine mai multor directoare, iar dacă directoarele aparţin mai multor utilizatori, acesta poate fi partajat.

Sub Linux există o structură predefinită de directoare, dintre care cele mai importante sunt:**.**
`/bin` conţine programe executabile (compilatoare, asambloare, interfeţe etc.
`/boot` conţine fişierul executabil de bootare al sistemului de operare.
`/dev` conţine conţine fişierele speciale de dispozitiv.
`/etc` conţine fişiere destinate configurării sistemului (conţine informaţii similare celor din Control Panel din Windows); dintre acestea amintim:
`/home` conţine directoarele gazdă ale utilizatorilor.
`/lib` conţine biblioteca cu fişiere utilizate atât de sistem cât şi de utilizatori.
`/lost+found` conţine fişierele salvate când apare o cădere a sistemului.
`/mnt` este punctul de montare standard pentru alte sisteme de fişiere.
`/proc` conţine un sistem de fişiere virtual. El nu există pe disc, fiind creat de nucleu în memorie. Este folosit pentru a păstra informaţii despre sistem, dintre care amintim:
`/sbin` conţine comenzi de administrare, care pot fi folosite de utilizatorii obisnuiti, numai dacă li se permite.
`/tmp` conţine fisiere de manevră folosite de utilitarele de sistem, fiind şterse atunci când nu mai sunt necesare.
`/usr` conţine programe, biblioteci, documentatii etc. pentru toti utilizatorii.
`/var` conţine fişiere temporare create de utilizatori, cum ar fi fişierele de log, cele descărcate de pe Internet, imaginea unui CD înainte de creare.

Partiţii şi blocuri. Un sistem de fişiere Linux este găzduit fie pe un periferic oarecare (hard-disc, CD, dischetă etc.), fie pe o *partiţie* a unui hard-disc; atât partiţiilor, cât şi suporturilor fizice reale li se spune *discuri Linux*.
Un disc Linux conţine:
- un bloc de boot;
- un superbloc
- mai multe blocuri care conţin inod-uri;
- mai multe blocuri care conţin fişiere.

Administrarea sistemului de fişiere. În sistemele Linux, accesul la dispozitivele periferice se realizează în mod diferit de sistemele de tip Windows. Nu există volume separate de genul `A:`,`C:` etc ; orice astfel de dispozitv este integrat în sistemul de fişiere local printr-o operaţie numită **montare**. Montarea asociază unui director întregul sistem de fişiere aflat pe dispozitivul montat. Mecanismul de montare oferă posibilitatea de a avea o structură de directoare unitară, care grupează fişiere de pe mai multe partiţii sau discuri. Prin utilizarea sistemului de fişiere NFS (Network File System), această structură de directoare va putea conţine şi sisteme de fişiere de pe un alt calculator din reţea. Caracterul unitar este dat de faptul că la o structura arborescentă, se adaugă un subarbore în cadrul arborelui deja existent, legâdu-l de un nod ales de noi (în acest caz directorul în care îl montăm), care trebuie să fie vid înainte de montare.

Operaţia de montare are rolul de a face disponibil conţinutul unui sistem de fişiere, asimilându-l în cadrul structurii de directoare a sistemului. Un sistem de fişiere poate fi montat/demontat la/de la ierarhia sistemului. Partiţia rădăcină este întotdeauna montată la

pornirea sistemului. Este imposibilă demontarea partiției rădăcină în timpul funcționării sistemului.

Ierarhia de fişiere de pe o partiție poate fi montată în orice director al sistemului rădacină, acesta numindu-se punct de montare. După montare, directorul radăcină al sistemului de fişiere montat, înlocuieşte conținutul directorului unde a fost montat. Pentru identificarea dispozitivelor periferice, sistemele Linux folosesc intrări speciale în directoare, numite fişiere dispozitiv (device files). Fisierele speciale care indică unitați de disc sau partiții sunt folosite la montare. În general aceste fişiere se găsesc în directorul `/dev` şi au denumiri standardizate. **Numele de discuri** care pot fi utilizate sunt prezentate în tabelul 6.1.2.

Nume	Descriere
`/dev/hda`	Primul hard disc IDE din sistem (Integrated Drive Electronics) (omologul lui C: din DOS şi Windows), conectat la IDE ca master drive.
`/dev/hdb`	Al doilea hard disc IDE din sistem, conectat la IDE ca slave drive.
`/dev/hdc`	Primul hard disc, conectat la al doilea controller IDE ca master drive.
`/dev/hdd`	Al doilea hard disc, conectat la al doilea controller IDE ca slave drive.
`/dev/sda`	Primul disc SCSI (Small Computer System Interface).
`/dev/sdb`	Al doilea disc SCSI.
`/dev/fd0`	Primul floppy disc (A: din DOS).
`/dev/fd1`	Al doilea floppy disc (B:din DOS).
`/dev/lp0`	Primul port paralel GNU/Linux
`/dev/lp1`	Al doilea port paralel GNU/Linux
`/dev/ttyS0`	Port serial

Tabelul 6.1.2. Nume de discuri.

Observații.
1. Hard discurile pot fi partiționate; prima partiție din a discului `hda` este `hda1`, a doua este `hda2` ş.a.m.d.
2. În general, avem următoarele denumiri standardizate (tabelul 6.1.3.).

Nume	Descriere
`fdX`	Unități de floppy
`hdX`	Unități HDD sau CDROM pe IDE .
`cdromX`	Unități cdrom (în general, este o legatură simbolică).
`scdX`	Discuri SCSI sau unități CDROM emulate SCSI sau pe USB.
`sdaX`	Unități de stocare pe USB (HDD-uri , ZIP-uri , FDD-uri, Card Readere, Flash-uri)

Tabelul 6.1.3. Denumiri standardizate de discuri.

Exemplu. `X` -urile de mai sus sunt numere corespunzătoare unității respective. Presupunem că avem două hard-disk-uri IDE, care vor fi identificate de Linux ca fiind `hda` şi `hdb`. În Windows ne-am referit la o partiție ca fiind `C:`, `D:`, `E:` etc. În Linux, dacă vrem să ne referim la partiția a 3-a de pe hard-disk-ul IDE slave, de pe controler-ul IDE primar, vom folosi `hdb3` ; `hd` se referă la tipul unitații, `b` se referă poziția unității, iar 3 este numărul partiției. Pentru aflarea partițiilor de pe sistem şi a tipului lor se foloseşte comanda `fdisk -l`

Partițiile specificate în fişierul `fstab`, pe care îl vom studia imediat devin stabile şi pot fi folosite în formele prescurtate ale comenzii `mount`. Sintaxa comenzii `mount` este de forma:
`mount [opțiuni] [<dispozitiv>] [director]`

Opțiunile sunt:
`-a` montează toate intrările din fstab;
`-t tip_sist_fis` specifică tipul sistemului de fișiere care este montat;
`-o optiuni` specifică opțiunile procesului de montare (de obicei sunt specifice tipului de sistem de fișiere). În această situație, opțiunile cele mai utilizate sunt:
`ro` montează sistemul de fișiere numai pentru citire (read-only);
`rw` montează sistemul de fișiere pentru citire și scriere (read-write implicit);
`exec` permite execuția fișierelor binare (implicit);
`loop` permite montarea unui fișier ca și când ar fi un dispozitiv.
Observație. Este permisă montarea unei imagini ISO a unui CD fără a scrie pe un CD-ROM.
În general, comanda `mount` este utilizată în două moduri:
- pentru a afișa discurile, partițiile și sistemele de fișiere de la distanță, care sunt montate curent (se tastează `mount` fără parametri);
- pentru a monta temporar un sistem de fișiere.

Exemplu.
`#mount`
va afișa toate sistemele de fișiere montate în sistem.

Exemplu.
`#mount -t ext3`
afișează toate sistemele de fișiere de tip ext3.

Exemplu.
`#mount -t ext3 -l`
listează etichetele partițiilor montate.

Exemplu.
La încărcarea sistemului de operare se execută automat:
`#mount -a`
prin care se montează simultan toate sistemele de fișiere configurate în `fstab`.

Pentru montarea altor sisteme de fișiere este indicat să se creze câte un director separat pentru fiecare dintre ele; se poate crea directorul `/mnt` și apoi în acest director să se creeze subdirectoare separate:
`/mnt/windows` – partitia pe care se afla Windows
`/mnt/cdrom` – unitatea de cdrom
`/mnt/floppy` – discheta de 1.44

Exemple.
Se tastează comenzile:
`#mount /dev/hda -t vfat /mnt/windows`, pentru a monta partiția de Windows
`#mount /dev/cdrom -r /mnt/cdrom` , pentru a monta un CD
`#mount /dev/fd0 -t /mnt/floppy` , pentru a monta o dischetă

Dupa teminarea utilizării unui sistem de fișiere montat parțial sau pentru demontarea temporară a unui sistem de fișiere se folosește comanda `umount`, a cărei sintaxă este:

```
#umount [optiuni] director
```

în care `director` este directorul care urmează a fi demontat.

Exemple. Pentru demontarea dischetei montată în exemplul anterior, se tastează:
`#umount /mnt/floppy`
Acelaşi efect se obţine dacă se tastează:
`#umount /dev/fd0`

Scrieţi o secvenţă de comenzi, prin care să se facă montarea unui sistem de fişiere aflat pe un `cdrom`, într-un subdirector al directorului gazdă.

Observaţii. 1. Este indicat să se folosească prima variantă (numele de director), deorece demontarea va eşua dacă unitatea este montată în mai multe locuri.
2. Dacă este afişat mesajul **device is busy**, înseamnă că demontarea a eşuat datorită faptului că un proces are un fişier deschis pe directoul (unitatea) care este obiectul demontării sau există un fişier de comenzi, al cărui director curent este cel care se doreşte a fi demontat.
3. Dacă se foloseşte `umount -l`, demontarea se va realiza de îndată ce unitatea va deveni liberă. Pentru a se forţa demontarea unui sistem de fişiere care nu este disponibil se foloseşte `umount -f`.

Utilizarea fişierului `fstab` pentru definirea sistemelor de fişiere utilizate. `fstab` este un fişier de configurare, ce conţine informaţii despre toate partiţiile şi dispozitivele de stocare ale calculatorului, precum şi cele legate de locul unde acestea ar trebui montate. Acest fişier este localizat în directorul `/etc`. Dacă nu se poate accesa partiţia Windows din Linux, nu se poate monta unitatea CD-ROM sau cea de floppy, atunci cauza probabilă este că fişierul `/etc/fstab` este configurat necorespunzator. `/etc/fstab` este un fişier text, deci se poate deschide şi edita cu orice editor de text din Linux, numai de utilizatorul cu drepturi de root. De asemenea, fişierul `fstab` este folosit pentru a monta la pornirea sistemului toate partiţiile configurate. Pe fiecare sistem există un fişier `/etc/fstab` cu un conţinut specific, datorită partiţiilor, dispozitivelor de stocare şi proprietatilor lor, ce sunt diferite de la un sistem la altul. În schimb, structura de bază a fişierului `fstab` este tot timpul aceeaşi.

Structura fişierului `fstab`. Fiecare linie conţine informaţii despre un dispozitiv sau o partiţie.
- Prima coloană conţine numele dispozitivului ce reprezintă sistemul de fişiere. Cuvântul `none` indică sisteme de fişiere (`/proc`, `/dev/shm`, `/dev/pts`) care nu sunt asociate cu dispozitive speciale. Cu opţiunea `label` se poate indica o etichetă de volum în locul unui nume de dispozitiv. În felul acesta, se poate muta un volum pe un alt nume de dispozitiv, fără a se face modificări în `fstab`.
- A doua coloană conţine punctul de montare în sistemul de fişiere. Trebuie ca punctul de montare să fie un director care există în sistem, altfel el trebuie creat manual. Unele partiţii şi dispozitive sunt montate automat la bootarea sistemului Linux.
- A treia coloană conţine tipul sistemului de fisiere.
- A patra coloană conţine optiunile de montare (comanda `mount`). Dacă partiţia respectivă nu se montează la bootare, se specifică opţiunea `noauto`.
- A cincea coloană (un număr) este folosită de comanda `dump` pentru a determina care sistem de fişiere trebuie salvat.
A şasea coloană (un număr) este folosită de comanda `fsck` pentru a determina ordinea în care va verifica sistemele de fişiere la rebootare. Sistemul pe care se găseste `/` va avea valoarea `1` în acest câmp, iar celelalte vor avea în general valoare `2` (sau `0` în cazul în care nu se doreşte verificarea lor).

Exemplu de fişier `fstab`

```
LABEL=/      /           ext3    defaults          1 1
LABEL=/boot  /boot       ext3    defaults          1 2
none         /dev/pts    devpts  gid=5,mode=620    0 0
/dev/fd0     /mnt/floppy auto    noauto,owner      0 0
none         /proc       proc    defaults          0 0
/dev/hda5    swap        swap    defaults          0 0
/dev/cdrom  /mnt/cdrom   iso9660 noauto,owner,
                                 kudzu,ro          0 0
/dev/hda1   /mnt/win     vfat    noauto            0 0
```

Adăugaţi o linie la fişierul `fstab` prin care să se monteze la boot-are partiţia a 3-a de pe hard-disk-ul IDE slave, cu verificarea sistemului de fişiere.

Observaţie. Partiţiile hard-disc de pe rădăcină, de încărcare (`/boot`), sunt montate la încărcarea sistemului de operare.

Crearea unui sistem de fişiere se realizează cu comanda `mkfs`, a cărei sintaxă este :

```
#mkfs -t <tip_sist_de_fis> <dispozitiv>
```

Exemple

```
#mkfs -t ext3 /dev/fd0
```

Crează un sistem de fişiere `ext3` pe dischetă.

```
#mkfs -t ext3 /dev/hda2
```

Crează un sistem de fişiere `ext3` pe a doua partiţie a discului `hda`.

Creaţi un sistem de fişiere pe un `cdrom`.

Afişarea informaţiilor despre partiţii se face cu comanda `df`. Comanda afişează informaţii despre partiţiile active din sistem cu excepţia celei de swap. Foloseşte opţiunea −h (human readable). Sunt incluse şi partiţiile de pe servere din reţea.

Exemplu.

```
df -h
Filesystem Size  Used  Avail Use% Mounted on
/dev/hda8  496M  183M  288M  39%  /
/dev/hda1  124M  8.4M  109M  8%   /boot
/dev/hda5  19G   15G   2.7G  85% /opt
/dev/hda6  7.0G  5.4G  1.2G  81% /usr
/dev/hda7  3.7G  2.7G  867M  77% /var
fs1:/home  8.9G  3.7G  4.7G  44% /.automount/fs1/root/home
```

Observăm că se listează un antet care conţine mai multe câmpuri, care indică tipul informaţiei de pe coloana respectivă; pe ultima linie sunt trecute informaţiile despre un sistem de fişiere montat pe un server din reţea.

Schema de alocare a blocurilor disc pentru un fişier. Fiecare sistem de fişiere Linux are câteva constante proprii, printre care amintim:

- lungimea unui inod;

- lungimea unui bloc;
- lungimea unei adrese disc (implicit câte adrese disc încap într-un bloc);
- câte adrese de prime blocuri din fişier se înregistrează direct în inod;
- câte referinţe se trec în lista de referinţe indirecte.

În cele ce urmează, pentru înţelegerea noţiunilor, vom lucra cu următoarele setări, specifice sistemelor Unix şi Linux mai vechi:
- un inod se reprezintă pe 64 octeţi;
- un bloc are lungimea de 512 octeţi;
- adresa disc se reprezintă pe 4 octeţi, deci încap 128 adrese disc într-un bloc;
- în inod se trec direct primele 10 adrese de blocuri;
- lista de adrese indirecte are 3 elemente.

Indiferent de valorile acestor constante, principiile după care se realizează alocarea de spaţiu, scrierea informaţiilor în fişier şi regăsirea informaţiilor sunt aceleaşi. Cu aceste constante, în figura 6.1.4. este prezentată structura pointerilor spre blocurile ataşate unui fişier Linux.

În inodul fişierului se află o listă cu 13 intrări, care desemnează blocurile fizice aparţinând fişierului. Primele 10 intrări conţin *adresele primelor* 10 blocuri de câte 512 octeţi care aparţin fişierului.

Intrarea numărul 11 conţine adresa *blocului de indirectare simplă*. El conţine adresele următoarelor 128 blocuri de câte 512 octeţi, care aparţin fişierului.

Intrarea numărul 12 conţine adresa *blocului de indirectare dublă*. El conţine adresele a 128 blocuri de indirectare simplă, care la rândul lor conţin fiecare, adresele a câte 128 blocuri, de 512 octeţi, cu informaţii aparţinând fişierului.

Intrarea numărul 13 conţine adresa *blocului de indirectare triplă*. În acest bloc sunt conţinute adresele a 128 blocuri de indirectare dublă, fiecare dintre acestea conţinând adresele a câte 128 blocuri de indirectare simplă, care la rândul lor conţin adresele a câte 128 blocuri, de câte 512 octeţi, cu informaţii ale fişierului.

Observaţie. Schema de alocare prezentată are drept caracteristică principală flexibilitatea ei, în sensul că se foloseşte un anumit mod de indirectare în funcţie de dimensiunea fişierului. Astfel, pentru fişiere de dimensiuni mai mici, nu se recurge la anumite scheme de indirectare, care presupun accesări ale discului.

În tabelul 6.1.4. sunt prezentate relaţiile dintre lungimea maximă a fişierului, numărul de accesări indirecte şi numărul total de accesări. Setările anterioare au fost utilizate de sistemele Unix/Linux mai vechi. La sistemele Linux actuale lungimea unui bloc este de 4096 octeţi care poate înregistra 1024 adrese, iar în inod se înregistrează direct adresele primelor 12 blocuri. În aceste condiţii, tabelul 6.1.4 se transformă în tabelul 6.1.5.

Observaţie. Sub Linux mai există şi alte sisteme de fişiere:
- *ReiserFS* este un sistem de fisiere cu jurnalizare. Acesta lucrează folosind metadate particulare asociate fişierelor, ceea ce îi permite să recupereze fisierele, după eventualele blocaje de sistem, cu o rapiditate şi o fiabilitate superioară altor sisteme. *ReiserFS* este disponibil începând cu versiunea 2.4.1 a nucleuluilui şi este folosit de multe distribuţii (printre care şi Gentoo – www.gentoo.org) ca sistem de fisiere implicit în locul clasicului *ext3*. Avantajul acestui sistem de fişiere constă în faptul că nu este legat de tehnologii anterioare, cum este cazul lui *ext3*.
- *iso9660* recunoaşte nume lungi de fişiere şi informaţii obţinute în stil UNIX (permisiuni pentru fişiere, proprietatea asupra fişierelor şi legături la fişiere).

- *proc* nu este un sistem de fişiere adevărat, ci o interfață de sistem de fişiere la nucleul Linux. Multe utilitare Linux se bazează pe *proc*.
- *swap* este utilizat pentru partițiile de memorie virtuală.
- *nfs* (Network File System) este utilizat pentru montarea sistemelor de fişiere de pe alte calculatoare din rețea.

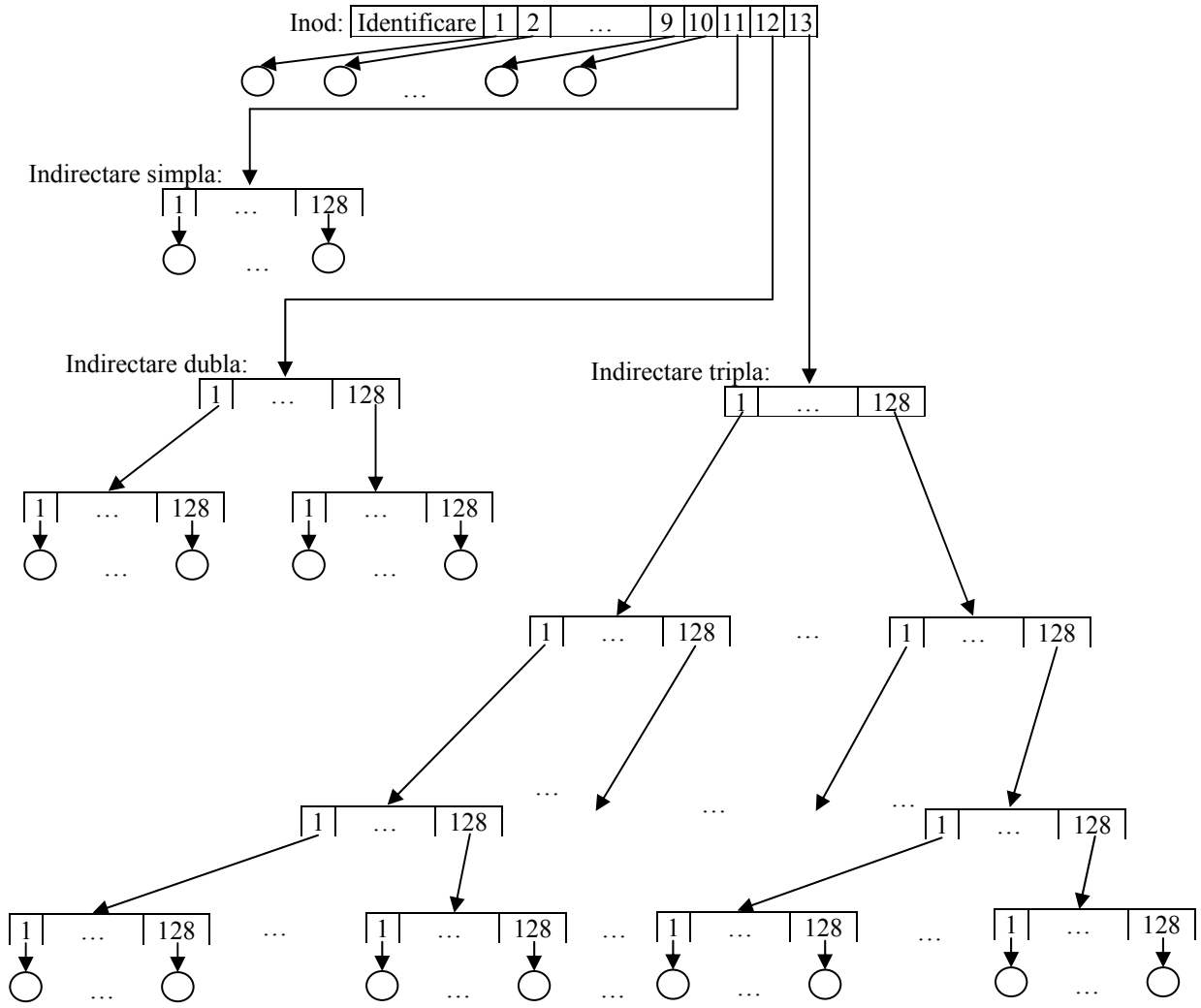


Figura 6.1.2. Structura pointerilor spre blocurile ataşate unui fişier Linux

Lungime maxima (blocuri)	Lungime maxima (octeți)	Accese indirecte	Accese la informații	Total accese
10	5120	-	1	1
10+128	70656	1	1	2
10+128+ +128^2=16522	8459264	2	1	3
10+128+128^2+ +128^3=2113674	1082201088	3	1	4

Tabelul 6.1.4. Legătura dintre dimensiunea fişierului şi numărul de accesări ale discului

Lungime maxima (blocuri)	Lungime maxima (octeți)	Accese indir.	Accese la inform.	Total acc.
12	49152	-	1	1
12+1024=1036	4243456	1	1	2
$12+1024+1024^2=$ 1049612	4299210752	2	1	3
$12+1024+1024^2+$ $+1024^3=1073741824$	4398046511104 (peste 5000Go)	3	1	4

Tabelul 6.1.5. Legătura dintre dimensiunea fişierului şi numărul de accesări ale discului dacă se măreşte dimensiunea blocului

Exemplu. Presupunem că avem un fişier de lungime `10` Mo. Ne propunem să determinăm ce tip de indirectare se foloseşte pentru alocarea de spaţiu pentru acest fişier, dacă se foloseşte prima schemă de alocare. Observăm că:

`10 Mo=10x1024x1024 octeţi=10x2`^{20} `octeti`

`1 bloc=512 octeti=2`^{9} `octeţi`

După alocare directă mai rămân:

`10x2`^{20}`-10x2`^{9} `octeţi = 10x2`^{9}`x2043 octeţi`

După indirectare simplă mai ramân:

`10x2`^{9}`x2043-128x512 octeţi = 2`^{9}`x20302 octeţi`

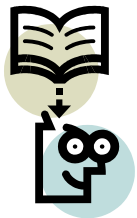
Prin redirectare dublă se mai poate aloca spaţiu pentru:

`128x128x512 octeţi=2`^{9}`x16384 octeţi.`

Deoarece `20302>16384,` rezultă că este necesară indirectare triplă.

Să ne reamintim...

Sub Linux nu există noţiune a de unitate; acestea sunt înlocuite de fişierele speciale. În sistemele de fişiere Linux, avem o structură arborescentă de fişiere, care se defineşte asemănător cu structura din sistemele MSDOS şi Windows. În cazul Linux, pentru directorul rădăcină şi separatorul de cale se foloseşte caracterul /. Sub Linux, există un director gazdă al fiecărui director, notat cu ~. De asemenea, există o structură predefinită de fişiere şi directoare.



Sub sistemele Unix/Linux nu există noţiunea de identificator de dispozitiv. Fiecare disc logic (partiţie) este văzut ca un fişier special.

Pentru ca într-un sistem de fişiere să se integreze un alt sistem de fişiere, de pe un alt suport, chiar de pe un alt calculator se foloseşte operaţia de montare. Operaţia inversă este operaţia de demontare. Cele două operaţii se pot realiza prin comenzi lansate de administratorul de sistem. Afişarea informaţiilor despre partiţii se face cu comanda `df`. Crearea unui sistem de fişiere se realizează cu comanda `mkfs`.

`fstab` este un fişier de configurare, ce conţine informaţii despre toate partiţiile şi dispozitivele de stocare ale calculatorului, precum şi cele legate de locul unde acestea ar trebui montate.

Schema de alocare a spaţiului pe disc pentru fişiere sub sistemele de Linux şi Unix, foloseşte anumite setări care ţin cont de caracteristicile hardware a sistemului de calcul respectiv.Schema este flexibilă, deoarece tipul de indirectare utilizat depinde de dimensiunea fişierului. Dacă aceasta este mică, blocurile pot fi accesate direct, prin pointeri care sunt memoraţi în inodul fişierului respectiv.

I. Întrebări.
1.În care sub-director predefinit se crează directorul gazdă al unui utilizator loginat.
2.Care dintre sub-directoarele predefinite conține un sistem de fișiere virtual.
3. În ce subdirectoare sunt plasate fișierele executabile.
4. Cum ne referim la partiția a doua de pe hard-disk-ul IDE slave, de pe controler-ul IDE primar.
5. În cazul căror sisteme de fișiere se folosește pentru modificare comanda `e2fsck`
6. În care director se salvează segmentele de fișier recuperate de `fsck` pe care nu le poate reunifica cu fișierul din care provin.

II. Determinați tipul de indirectare folosit pentru un fișier a cărui dimensiune este de `100` MO, folosind cele două setări ale schemelor de alocare. Specificați câte blocuri sunt ocupate prin ultima indirectare utilizată.

M6.U1.6. Administrarea proceselor

Stările unui proces şi tranziția între stări. Un proces sub Linux se poate afla într-una dintre următoarele stări:
`New`: procesul este creat, dar încă nu este gata de execuție.
`ReadyMemory`: procesul ocupă loc în memoria RAM şi este gata pentru a fi executat.
`ReadySwapped`: procesul ocupă spațiu numai în memoria secundară (memoria virtuală sau zona swap pe disc), dar este gata de execuție.
`RunKernel`: instrucțiunile procesului sunt executate în mod nucleu sau protejat; astfel de instrucțiuni corespund unui apel sistem, unui handler de întreruperi etc.
`RunUser`: sunt executate instrucțiunile cod maşină ale procesului în mod utilizator.
`Preempted`:corespunde unui proces care a fost forțat de către un alt proces cu o prioritate superioară.
`SleepMemory`: procesul se află în memorie, dar nu poate fi executat până când nu se produce un eveniment care să-1 scoată din această stare.
`SleepSwapped`: procesul este evacuat pe disc şi el nu poate fi executat până când nu se produce un eveniment care să-1 scoată din această stare.
`Zombie`: s-a terminat execuția procesului, înaintea execuției procesului părinte (noțiune care o vom discuta în mai târziu).
În figura 6.1.3 este prezentată diagrama stărilor în care se poate afla un proces şi a tranzițiilor posibile între aceste stări.

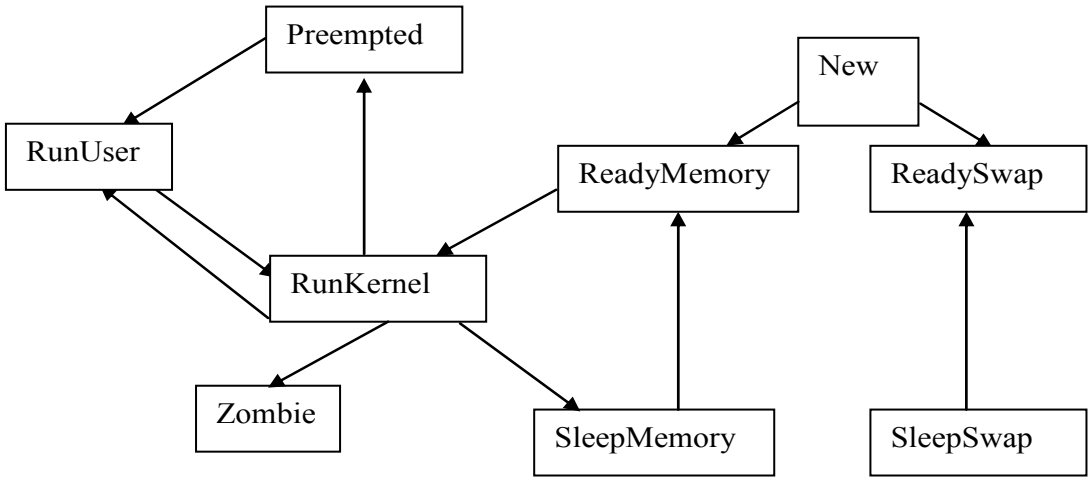


Figura 6.1.3 Diagrama stărilor şi a tranzițiilor între stări ale unui proces.

Tranzițiile posibile sunt:
New→ReadyMemory: există suficientă memorie internă pentru a fi alocată procesului.
New→ReadySwap: Nu există suficientă memorie internă pentru a fi alocată procesului şi procesului i se alocă numai spațiu în memoria swap (virtuală).
RunUser→RunKernel: Procesul cere execuția unui apel de sistem.
RunKernel→RunUser: Revenire dintr-o operație executată în mod protejat.
RunKernel→Zombie: Se aşteaptă terminarea procesului părinte, pentru a se actualiza anumite statistici; procesul respectiv nu mai are alocate resurse, el există numai în tabela cu procese a sistemului.
RunKernel→Preempted: Trecerea în starea Preempted apare atunci când în sistem se impune execuția unui proces cu o prioritate mai înaltă; acest lucru este decis de nucleu, deoarece nu poate fi forțat un proces care execută o operație în mod nucleu.
Preempted→RunUser: Când procesul aflat în starea Preempted are prioritatea cea mai înaltă, este trecut în RunUser.
RunKernel→SleepMemory: Se aşteaptă apariția unui eveniment ca să se continue execuția procesului.
SleepMemory→ReadyMemory: Procesul a fost trezit de evenimentul corespunzător şi concurează pentru a fi servit de CPU.
SleepMemory→SleepSwapped: Pentru procesul respectiv, nu mai există suficientă memorie internă, aşa că este trecut în memoria swapp.
SleepSwapped→ReadySwap: S-a întâmplat evenimentul necesar trezirii procesului, dar nu există suficientă memorie internă pentru acesta.

Ierarhia de procese. Fiecare proces este reprezentat printr-un identificator unic numit PID (Process Identification). Procesul cu pid-ul 0 (*idle process)* este procesul pe care nucleul îl execută când în sistem nu sunt procese executabile. Procesul cu pid-ul 1 este procesul *init*, invocat de nucleu la sfârşitul procedurii de încărcare a sistemului de operare. În afara cazului în care utilizatorul specifică explicit nucleului care este procesul pe care să-l execute (prin intermediul comenzii init), nucleul trebuie să identifice un proces corespunzător (init) al cărui proprietar este. Nucleul Linux încearcă unul dintre următoarele procese, în următoarea ordine: /sbin/init, /etc/init, /bin/init, /bin/sh. Dacă nici unul dintre aceste patru procese nu poate fi executat, nucleul Linux intră într-o stare de aşteptare. Dupa găsirea de către nucleu, procesul init manipulează restul proceselor de boot-are.

Exemplu. Acțiuni realizate de procesul init sunt inițializarea sistemului, lansarea unor servicii, lansarea programului de loginare etc.

Alocarea PID-ului. Implicit, nucleul impune o valoare maximă a PID-ului, care este 32768 (pentru compatibilitate cu sistemele Unix mai vechi, care au functionat pe un procesor pe 16 biți). Administratorul de sistem poate seta valoarea maxima prin programul /proc/sys/kernel/pid_max. Valorile PID se alocă secvențial de către sistemul de operare; cele alocate anterior nu pot fi reutilizate.

Cu excepția procesului init, fiecare proces are un tată (procesul în interiorul căruia este declarat); procesul respectiv este un proces fiu. În cadrul procesului fiu, este recunoscut identificatorul procesului părinte (PPID). Fiecare proces aparține unui *utilizator(user)* şi unui *grup de utilizatori(group)*. Această relație este utilizată pentru a controla drepturile de acces. Aceste valori sunt numere întregi corespunzătoare unor identificatori specificați in fişierele /etc/passwd şi /etc/group. Fiecare proces fiu moşteneste *user*-ul şi *group*-ul tatălui.De asemenea, fiecare proces face parte dintr-un *grup de procese*, care exprimă o

relație cu alte procese. Procesele fiu şi tată aparțin aceluiaşi grup. Din perspectiva utlizatorului, un grup de procese formează un *job*.

Exemplu. Două sau mai multe procese legate prin pipe, formează un grup de procese.

Planificarea sub linux. Sub Linux procesele/firele de execuție se împart în `3` clase de priorități:

- în timp real, planificate după strategia FIFO;
- în timp real, planificate după strategia round-robin;
- cu partajarea timpului.

Procesele FIFO în timp real au cea mai mare prioritate şi nu pot fi forțate. Procesele round-robin în timp real se execută după metoda round-robin. Fiecare proces are o prioritate de execuție, a cărei valoare implicită este `20`; această prioritate poate fi modificată folosind comanda `nice valoare`, prin care prioritatea devine `20-valoare`. Parametrul `valoare` este un număr întreg din mulțimea $\{i / i \in N, -20 \le i \le 19\}$, deci prioritățile sunt din mulțimea $\{i / i \in N, 1 \le i \le 40\}$. Scopul acestei metode, este de a face importanța serviciilor oferite de Linux proporțională cu prioritatea; procesele cu prioritate mai mare primesc un timp de răspuns mai rapid şi un timp CPU mai mare. În plus față de prioritate, fiecare proces are asociată o cuantă de timp, în cadrul planificării execuției după metoda round-robin. Cuanta reprezintă un număr de tacte ale ceasului sistemului, cât CPU este alocată procesului. Dacă ceasul funcționează la 100 Hz, fiecare tact este de `10` msec., ceea ce se numeşte **jiffy(moment).**

În cele ce urmează, este prezentat modul de utilizare de către planificator a priorității şi cuantei. Mai întâi se determină valoarea `goodness` (cât de bun) a fiecărui proces gata de execuție, aplicând următoarele reguli:

```
if (class == real_time) goodness = 100 + priority;
if (class == timesharing&& quantum > 0)
    goodness=quantum+priority;
if (class == timesharing && quantum == 0) g  goodness = 0;
```

Ambele clase de procese în timp real se încadrează în regula dată de primul `if`. Fiecare astfel de proces primeşte o valoare `goodness` mai mare decât a tuturor proceselor. Dacă procesului care a fost ultimul servit de CPU, i-a mai rămas o parte din cuanta de timp, acesta primeşte o valoare `goodness` mai mare, astfel încât dintre mai multe procese care au aceeaşi valoare a priorității, acesta va fi selectat pentru a fi executat, dacă procesul respectiv este gata de execuție.

Algoritmul de planificare se deruleaza astfel:

1. Se selectează procesul cu valoarea `goodness` cea mai mare.
2. În timpul execuției procesului, cuanta sa este decrementata cu `1` la fiecare tact de ceas.
3. CPU este luat procesului, dacă se întîmplă oricare dintre condițiile următoare:
   - Cuanta sa ajunge la `0`;
   - Procesul intră în starea de blocare;
   - Un proces blocat anterior cu o valoare `goodness` mai mare devine gata de execuție.

Cuantele proceselor tind să devină `0`. Dacă proceselor care urmează să efectueze o operație de I/O le mai ramâne o parte din cuantă alocată, planificatorul resetează toate valorile `Quantum`, conform formulei: `quantum = (quantum/2) + priority`

Să ne reamintim...

Un proces sub Linux se poate afla într-una dintre următoarele stări:`Created, ReadyMemory,ReadySwapped,RunKernel,RunUser,Preempted, ReadyMemory,RunUser,SleepMemory,SleepSwapped,Zombie.`

Fiecare proces este reprezentat printr-un identificator unic numit PID , alocat de către nucleul Linux. Procesul cu pid-ul `0` este procesul pe care nucleul îl execută când în sistem nu sunt procese executabile. Procesul cu pid-ul `1` este procesul *init*, invocat de nucleu la sfârşitul procedurii de încărcare a sistemului de operare.

Cu excepția procesului `init`, fiecare proces are un tată (procesul în interiorul căruia este declarat); procesul respectiv este un proces fiu. În cadrul procesului fiu, este recunoscut identificatorul procesului părinte (PPID). Fiecare proces aparține unui *user* şi unui *group*. Această relație este utilizata pentru a controla drepturile de acces. Fiecare proces fiu moşteneste *user*-ul şi *group*-ul tatălui.
Sub Linux procesele/firele de execuție se împart în `3` clase de priorități: în timp real, planificate după strategia FIFO; în timp real, planificate după strategia round-robin; cu partajarea timpului. Procesele FIFO în timp real au cea mai mare prioritate şi nu pot fi forțate. Procesele round-robin în timp real se execută dupa metoda round-robin. Fiecare proces are o prioritate de execuție, care poate fi modificată folosind comanda `nice.`

I.Întrebări.
1.Care este diferența dintre stările `ReadyMemory` şi `ReadySwapped.`
2. Care este diferența dintre stările `RunKernel` şi `RunUser`.

3. Care este diferența dintre stările `SleepMemory` şi `SleepSwapped.`
4. Prin ce program administratorul de sistem poate seta valoarea maxima a PID-ului.
5. Cum selectează nucleul Linux procesul `init.`
6. Ce relație este utilizată pentru a controla drepturile de acces.
Care este în diferența între procesele în timp real, planificate după strategia FIFO şi cele în timp real, planificate după strategia round-robin.
7. Cum utilizează planificatorul prioritatea şi cuanta.
8. Ce se întâmplă cu cuanta de timp, dacă procesele care urmează să efectueze o operație de I/O le mai ramâne o parte din cuantă alocată.

M6.U1.7. Gestiunea memoriei sub Linux

Structura spațiului de memorie alocat unui proces. Fiecare proces are alocat un spațiu de adrese din memoria internă şi unul din memoria virtuală. Procesul accesează memoria internă în două moduri: **utilizator** şi **nucleu.** Porțiunea accesată în mod **utilizator** este formată din:
- Partea de **instrucțiuni** cod maşină a procesului, care poate fi partajată cu alte procese.
- Datele inițializate **numai pentru citire (**constantele utilizate de către proces).
- Datele inițializate pentru **citire** şi **scriere (**variabilele procesului inițializate la compilare).
- Date neinițializate este zona ocupată de către restul variabilelor, cu excepția celor cărora li se alocă spațiu pe stivă sau în zona heap.
-Heap este zona unde se alocă spațiu pentru variabilele dinamice.

- Stiva este zona de memorie folosită pentru transferul parametrilor, în cazul apelului de funcții.
- Tabela proceselor conține câte o intrare pentru fiecare proces;
- Tabela paginilor de memorie virtuală coține câte o intrare pentru fiecare pagină din memorie, în care sunt păstrate diverse informații, printre care: procesul care foloseşte pagina respectivă, drepturi de acces etc. În cazul sistemelor Linux care folosesc o schemă de alocare segmentată şi paginată, mai există şi o tabelă de translatare segment - pagină de memorie virtuală.
- Tabela descriptorilor de fişiere deschise din sistem conține câte o intrare pentru fiecare descriptor deschis.
- Tabela inodurilor conține câte o intrare pentru fiecare i-nod de pe disc.
Observație. Conținutul primelor trei zone corespunde fişierului executabil corespunzător procesului. Pe lângă acestea, un fişier executabil mai conține şi alte informații care sunt folosite de către nucleul Linux la inițializarea modului de lucru protejat (nucleu).
Porțiunea utilizată în mod nucleu este administrată de către nucleul Linux şi este accesibilă procesului numai în modul de execuție protejat, prin apeluri de sistem specifice. Ea are o parte statică (de dimensiune fixă) şi una dinamică (de dimensiune variabilă). Această zonă conține:

- prioritatea procesului; un număr natural cuprins între 1 şi 39 folosit la planificarea execuției;
- semnalele netratate trimise procesului;
- statistici de timp (timpul de utilizare CPU, din momentul creării lui, folosit de algoritmul de planificare execuției);
- statutul memoriei procesului (imaginea procesului se află în memoria principală sau în memoria swap);
- pointer la următorul proces din coada de procese aflate în starea *Ready* (dacă procesul curent este în starea *Ready*);
- descriptorii evenimentelor care au avut loc cât timp procesul a fost în starea *Sleeping*.
- Zona User: este memorată în spațiul de adrese al procesului, dar este accesibilă numai din modul de execuție nucleu (intrarea în tabela proceselor se află în nucleu). Ea conține informații de control, care trebuie să fie accesibile nucleului sistemului de operare când se execută în contextul acestui proces şi anume:
  - pointer la intrarea în tabela proceselor care corespunde acestei zone;
  - UID, EUID, GID şi EGID - folosite la determinarea drepturilor de acces ale procesului;
  - timpul de execuție al procesului, atât în modul user, cât şi în modul nucleu;
  - vectorul acțiunilor de tratare a semnalelor; la recepționarea unui semnal, în funcție de valorile acestui vector, procesul poate să se termine, să ignore semnalul sau să execute o anumită funcție specificată de utilizator;
  - terminalul de control al procesului (cel de pe care a fost lansat în execuție, dacă există);
  - valorile returnate şi posibilele erori rezultate în urma efectuării unor apeluri sistem;
  - directorul curent şi directorul rădăcină;
  - zona variabilelor de mediu;
  - restricții impuse de nucleu procesului (dimensiunea procesului în memorie etc.);
  - tabela descriptorilor de fişiere conține datele relative la toate fişierele deschise de către proces;
  - masca drepturilor implicite de acces pentru fişiere nou create de către proces.
  - tabela regiunilor din memorie ale procesului este o tabelă de translatare a adreselor virtuale în adrese fizice pentru regiunile (secțiunile) programului, care este executat în contextul acestui proces;

Exemplu. Regiuni din memorie ale proceselor sunt secțiunea de text, de date neinițializate, de date inițializate citire/scriere.

- stiva nucleu este memoria alocată pentru stiva folosită de nucleul sistemului de operare în modul de execuție nucleu;

Exemplu. Aplicații care folosesc stiva nucleu sunt apelurile sistem.

- contextul registru (hardware) este o zonă de memorie unde se salvează conținutul regiştrilor generali, de stivă, de execuție, de control şi de stare în momentul în care algoritmul de planificare decide să dea controlul procesorului unui alt proces. Aceşti regiştrii ai procesorului se salvează pentru a putea relua execuția procesului, de unde s-a rămas.

Nucleul Linux trebuie să administreze toate aceste zone din memoria virtuală ale unui proces. În tabela procesului, există un pointer către o structură numită `mm_struct`. Această structură conține informații despre imaginea executabilă şi un pointer către tabelele cu paginile procesului. Ea conține pointeri către o listă cu elemente de tipul `vm_area_struct`; cele mai importante câmpuri ale structurii `vm_area_struct` sunt:
`vm_start`, `vm_end`; reprezintă adresa de început, respectiv de sfârşit ale zonei de memorie (aceste câmpuri apar în `/proc/*/maps`);
`vm_file`, pointer-ul la structura de fişiere asociată (dacă există);
`vm_pgoff`, deplasamentul zonei în cadrul fişierului;
`vm_flags`, un set de indicatori;
`vm_ops`, un set de funcții de lucru asupra acestei zone.
Toate structurile `vm_area_struct` corespunzătoare unui proces sunt legate printr-o listă; pentru ca regăsirea datelor să se facă mai rapid, ordinea în listă este odată de ordinea crescătoare a adreselor virtuale de unde încep zonele respective.
Zonele de memorie ale unui proces pot fi vizualizate inspectând `procfs`.

Exemplu.

```
$cat /proc/1/maps
08048000-0804f000 r-xp 00000000 03:01 401624 /sbin/init
0804f000-08050000 rw-p 00007000 03:01 401624 /sbin/init
08050000-08071000 rw-p 08050000 00:00 0
40000000-40016000 r-xp 00000000 03:01 369654 /lib/ld-2.3.2.so
40016000-40017000 rw-p 00015000 03:01 369654 /lib/ld-2.3.2.so
40017000-40018000 rw-p 40017000 00:00 0
4001d000-40147000 r-xp 00000000 03:01 371432 /lib/tls/libc-2.3.2.so
40147000-40150000 rw-p 00129000 03:01 371432 /lib/tls/libc-2.3.2.so
40150000-40153000 rw-p 40150000 00:00 0
bffff000-c0000000 rw-p bffff000 00:00 0
ffffe000-fffff000 ---p 00000000 00:00 0
```

Relocarea paginilor virtuale de memorie sub Linux. În cazul sistemelor de calcul cu procesor pe 64 de biți, o adresă virtuală este formată din `4` câmpuri: director global, director de mijloc, pagină şi deplasament. Astfel, pentru translatarea unei adrese virtuale într-una fizică se foloseşte o structură arborescentă pe `3` niveluri (figura 6.1.4). Fiecare proces are alocată o intrare în tabela director global. Această intrare conține un pointer către tabela director de mijloc şi în această tabelă există o intrare care conține un pointer către tabela de pagini; în această tabelă, există o intrare care conține un pointer către o pagină din memoria fizică, din care pe baza deplasamentului se obține o locație fizică în care este încărcată pagina

virtuală respectivă.
Observație. În cazul sistemelor care funcționează cu un microprocesor pe 32 de biți, funcționează o schemă pe două niveluri.

Algoritmul de înlocuire a paginilor. Fiecare pagină virtuală încărcată într-o pagină fizică alocată unui proces, are asociată o variabilă reprezentată pe `8` biți. De fiecare dată când se face referință la o pagină se incrementează variabila asociată. Aceste variabile, formează o listă. Periodic, sistemul parcurge această listă şi decrementează valorile acestor variabile.O pagină pentru care variabila asociată are valoarea `0`, este eligibilă pentru a fi înlocuită în momentul când se pune problema încărcării unei pagini virtuale. Dacă există mai multe pagini eligibile, se alege una dintre ele folosind regula FIFO. Această metodă este o variantă a algoritmului clasic LRU (**L**east **R**ecently **U**sed).
Observație. Linuxul admite existența modulelor încărcate dinamic, de obicei drivere de dispozitiv. Acestea pot fi de o dimensiune arbitrară şi necesită o zonă contiguă din memoria internă gestionată de nucleu. Pentru a se obține astfel de zone de memorie, se folosește metoda alocării de memorie prin camarazi.

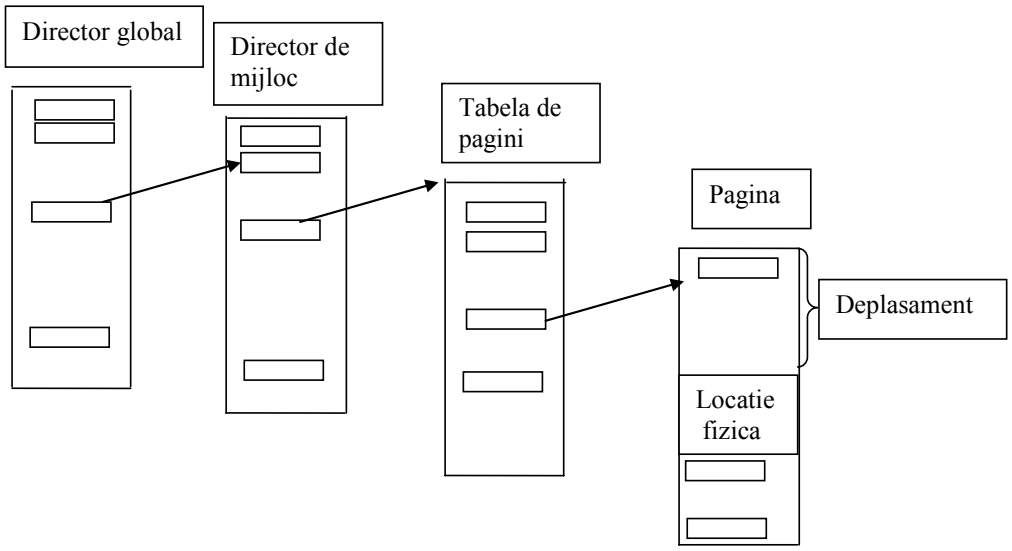


Figura 6.1.4. Organizarea arborescentă a schemei de translatare a paginilor virtuale

Să ne reamintim...
Fiecare proces are alocat un spațiu de adrese din memoria internă şi unul din memoria virtuală. Procesul accesează memoria internă în două moduri: **utilizator** şi **nucleu.**
Translatarea adreselor virtuale în adrese fizice se face, în cazul sistemelor de calcul pe 64 de biți pe baza unei structuri arborescente pe trei niveluri.
Algoritmul de înlocuire a paginilor se bazează pe selectarea paginii celei mai puțin utilizate.

Întrebări.
1.Conținutul căror zone corespunde fişierului executabil corespunzător procesului.
2.Care este zona de memorie folosită pentru transferul parametrilor, în cazul apelului de funcții.
3.Care zonă este memorată în spațiul de adrese al procesului, dar este accesibilă numai din modul de execuție nucleu.
4.Cum se selectează o pagină pentru a fi înlocuită.

M6.U1.8. Test de evaluare a cunoştinţelor
Marcaţi varianta corectă.

1. Serverul X rulează pe:

a) Calculatorul pe care se execută serverul de nume. ☐
b) Calculatorul pe care se execută serverul de samba. ☐
c) Calculatorul la care s-a conectat utilizatorul ☐
d) Calculatorul la care lucrează administratorul. ☐

2. Care dintre componentele sistemului de operare sunt scrise în limbajul de asamblare al calculatorului gazdă:

a) Nucleul ☐
b) Bibliootecile ☐
c) Driverele de memorie. ☐
d) Driverele de dispozitiv ☐

3. Pentru a afişa toate informaţiile din paginile de manual, în comanda `man` se foloseşte opţiunea:

a) -s ☐
b) -p ☐
c) -m ☐
d) -a ☐

4. Un i-nod reprezintă:

a) Descriptorul unui fişier ☐
b) Primul bloc al unui fişier ☐
c) Ultimul bloc al unui fişier ☐
d) Un nod de index al unui fişier ☐

5. Subdirectorul gazdă este asociat:

a) Unei gazde ☐
b) Unui utilizator ☐
c) Unui server ☐
d) Unui grup de utilizatori ☐

6.Simbolul ~ desemnează:

a) Sub-directorul gazdă al utilizatorului loginat ☐
b) Sub-directorul gazdă al grupului de utilizatori. ☐
c) Sub-directorul gazdă al administratorului ☐
d) Sub-directorul gazdă al utilizatorului cu drepturi de root. ☐

7. Fisierele speciale identifică:

a) Locaţii unde se salvează informaţiile despre utilizatori ☐
b) Locaţii unde se salvează informaţiile despre procese. ☐
c) Discuri ☐
d) Socketuri. ☐

8. `/dev/hdc2` însemnă:

a) A doua partiţie a primului hard disc, conectat la al doilea controller SCSI ca master drive. ☐
b) A doua partiţie a primului hard disc, conectat la al doilea controller IDE ca master drive. ☐
c) A doua partiţie a celui de-al treilea hard disc, conectat la al doilea controller SCSI ca master drive. ☐
d) A doua partiţie a celui de-al treilea hard disc, conectat la al doilea controller IDE ca master drive. ☐

9. Starea `Preempted`:

a) corespunde unui proces care a fost forţat de către un alt proces cu o prioritate inferioară. ☐
b) corespunde unui proces care a fost forţat de către un alt proces cu o prioritate superioară. ☐
c) corespunde unui proces care a fost evacuat din memorie. ☐
d) corespunde unui proces care aşteaptă să execute un apel de sistem. ☐

10. Heap este:

a) zona de memorie virtuală unde se alocă spațiu pentru variabilele statice. ☐

b) zona de memorie fizică unde se alocă spațiu pentru variabilele statice. ☐

c) zona de memorie virtuală unde se alocă spațiu pentru variabilele dinamice. ☐

d) zona de memorie fizică unde se alocă spațiu pentru variabilele dinamice. ☐

11. În cazul sistemelor de calcul cu procesor pe 64 de biți, o adresă virtuală este formată din 4 câmpuri:

a) director rădăcină, director de mijloc, pagină şi deplasament. ☐

b) director rădăcină, director de mijloc,segment şi deplasament. ☐

c) director global, director de mijloc, pagină şi deplasament. ☐

d) director global, director de mijloc, segment şi deplasament. ☐

12. Algoritmul de înlocuire a paginilor sub Linux o variantă a algoritmului clasic:

a) NRU ☐

b) LRU ☐

c) FIFO. ☐

d) Belady ☐

13. Care procese au cea mai mare prioritate şi nu pot fi forțate:

a) Driverele. ☐

b) Apelurile de sistem. ☐

c) Procesele round-robin în timp real ☐

d) Procesele FIFO în timp real ☐

14. Grupul de procese exprimă:

a) o relație între procese ☐

b) o relație între joburi ☐

c) o relație între procese şi utilizator. ☐

d) o relație între procese şi un grup de utilizatori. ☐

M6.U1.9. Rezumat.

Întreaga evoluție a sistemelor de operare Unix şi Linux este caracterizată de introducerea unor concepte noi. Astfel, chiar prima versiune a sistemului de operare UNIX a introdus un interpretor de comenzi care să fie un proces utilizator şi fiecare comandă să fie un proces; de asemenea a implementat un sistem ierarhic de organizare a fişierelor. Unele dintre marile firme de software şi marile universități americane şi-au adus contribuția la dezvoltarea Unix-ului. Un aspect important, este standardizarea diverselor versiuni ale sistemului Unix. Linux este un continuator al Unix, care a preluat facilitățile oferite utilizatorilor, dar spre deosebire de acesta este un produs liber, fiind pus la dispoziția tuturor celor interesați pe Internet.

Linux -ul este un sistem de operare multi-user şi multi-tasking ce oferă utilizatorilor numeroase utilitare interactive. Unix şi Linux au fost proiectate de programatori pentru programatori. Pornind de la acest fapt, interactivitatea şi facilitățile de dezvoltare ale programelor au fost elemente prioritare ale proiectanților sistemului. Interactivitatea sistemului este facilitată şi de interfața cu utilizatorul, care poate fi înlocuită sau dezvoltată, fiind ea însăşi un proces.

Interfața cu utilizatorii se poate face în mod text sau grafic. Terminalele virtuale sunt utilizate pentru introducerea de comenzi sub formă de texte. Sub Linux, atât comenzile cât şi interpretoarele de comenzi (shell) sunt procese care se execută în spațiul de memorie utilizator. Interfața grafică oferită de sistemele Linux se bazează pe **X Window System**; acesta este un cadru ce permite desenarea de ferestre pe un

afişaj în care imaginea este reprezentată ca o matrice de puncte.

O componentă esențială a sistemului Linux este nucleul, carea are o organizare monolitică; pentru a rezolva anumite necesități ale sistemului, el se execută ca unic proces. Orice modificare a nucleului, presupune o nouă editare de legături, reinstalarea şi, evident o nouă încărcare ale sistemului de operare.

Sistemele Linux conțin manuale interactive prin care sunt oferite informații despre comenzi. Utilizarea paginilor manualelor se face cu comanda `man`. Informațiile obținute cu comanda `info` sunt mai recente şi mai uşor de utilizat În multe cazuri, paginile din manuale fac trimitere la paginile info. Cu comanda `whatis` se pot obține explicații scurte despre comenzi. Cu opțiunea `–help`, care urmează o comandă, se obțin explicații scurte despre utilizarea comenzii şi o listă a opțiunilor disponibile.

Pe lângă fişierele obişnuite, privite ca şiruri de octeți, sistemul Linux utilizează şi alte tipuri de fişiere. Prin introducerea conceptului de *fişier special*, Linux priveşte fiecare dispozitiv de I/O ca şi un fişier de tip special. Se realizează astfel simplitatea şi eleganța comenzilor şi numele unui dispozitiv poate fi transmis ca argument.

Pe lângă structura arborescentă de directoare şi fişiere, întâlnită la alte sisteme de operare (MS-DOS, Windows), Linux introduce structura de **graf aciclic**. Astfel, un **fişier obişnuit** poate aparține mai multor directoare; acest lucru permite implementarea unui mecanism elegant de partajare a fişierelor.

Conceptul de montare permite conectarea unui sistem de fişiere, de pe un anumit disc, la un director existent pe sistemul de fişiere implicit. Acest lucru este necesar, în condițiile în care, spre deosebire de alte sisteme de operare ca(DOS, Windows, etc.) în specificarea fişierelor Linux *nu apare zona de periferic*(perifericele sunt fişiere speciale). Operația de *demontare* are efectul invers.

Linux foloseşte fişierul de configurare `fstab` care conține informații despre toate partițiile şi dispozitivele de stocare ale calculatorului, precum şi cele legate de locul unde acestea ar trebui montate. De asemenea, fişierul `fstab` este folosit pentru a se monta la pornirea sistemului toate partițiile configurate.

Linux propune o standardizare a organizarii unui disc. În acest sens, întâlnim blocul 0, care conține programul de încărcare al **sistemului de operare,** blocul 1 numit şi superbloc, care coține o serie de informații prin care se defineşte sistemul de fişiere de pe disc, blocuri care conțin informații despre fişierele memorate de pe disc şi blocurile in care se memorează conținuturile fişierelor.

În ceaa ce priveşte alocarea blocurilor disc pentru un fişier, linux propune o metodă flexibilă şi eficientă, care ia în considerare dimensiunea fişierelor şi realizează o regăsire rapidă a informațiilor stocate pe discuri.
Sistemele de operare Linux sunt caracterizate de conceptul de multiprogramare. În acest sens, conceptul de proces capătă un rol esențial. În funcție de cerințele procesului respectiv, de interacțiunea cu sistemul sau cu alte procese, un proces sub Linux se poate afla într-una dintre următoarele stări: `Created`, `ReadyMemory`, `ReadySwapped`, `RunKernel`, `RunUser`, `Preempted`, `ReadyMemory`, `RunUser`, `SleepMemory`, `SleepSwapped`, `Zombie`. De asemenea, tranzițiile dintr-o stare în alta oferă o imagine abstractă a execuției proceselor.

Fiecare proces are alocat un spațiu de adrese din memoria internă şi unul din memoria virtuală. Procesul accesează memoria internă în două moduri: **utilizator** şi **nucleu.** Translatarea adreselor virtuale în adrese fizice se face, în cazul sistemelor de calcul pe 64 de biți pe baza unei structuri arborescente pe trei niveluri.Algoritmul de înlocuire a paginilor se bazează pe selectarea paginii celei mai puțin utilizate.
Sub Linux procesele/firele de execuție se împart în `3` clase de priorități: în timp real, planificate după strategia FIFO; în timp real, planificate după strategia round-robin; cu partajarea timpului. Procesele FIFO în timp real au cea mai mare prioritate şi nu pot fi forțate. Procesele round-robin în timp real se execută dupa metoda round-robin.
Fiecare proces este reprezentat printr-un identificator unic numit PID , alocat de către nucleul Linux. Procesul cu pid-ul `0` este procesul pe care nucleul îl execută când în sistem nu sunt procese executabile. Procesul cu pid-ul `1` este procesul *init*, invocat de nucleu la sfârşitul procedurii de încărcare a sistemului de operare.

Cu excepția procesului `init,` fiecare proces are un tată (procesul în interiorul căruia este declarat); procesul respectiv este un proces fiu. În cadrul procesului fiu, este recunoscut identificatorul procesului părinte (PPID). Fiecare proces aparține unui *user* şi unui *group*. Această relație este utilizata pentru a controla drepturile de acces. Fiecare proces fiu moşteneste *user*-ul şi *group*-ul tatălui.

Semnalul este o întrerupere software pe care un proces o primeşte spre a fi informat despre apariția unui eveniment. Semnalele se folosesc pentru: situații de excepție, alarme, terminări neaşteptate şi pentru comunicația între procese. Semnalul poate fi privit şi ca o avertizare pe care o primeşte un anumit proces. Fiecare semnal are un nume care este prefixat de **SIG**. Numele semnalelor sunt definite prin constante simbolice care sunt numere întregi strict pozitive.

Unitatea de învățare M6.U2. Sistemele Windows

Cuprins

M6.U2.1. Introducere

Microsoft Windows este numele unei familii de sisteme de operare create de compania Microsoft. Progresul sistemelor Windows, este legată de apariția şi evoluția calculatoarele personale. Primele calculatoare personale au funcționat sub sistemele de operare DOS. MS-DOS a fost varianta Microsoft a sistemului DOS. Microsoft a introdus Windows pe piață, pentru prima dată în noiembrie 1985, ca o interfață grafică la sistemul de operare MS-DOS, deoarece interfețele grafice erau din ce in ce mai apreciate. Microsoft Windows a ajuns cu timpul să predomine pe piața de calculatoare mici, întrecând Mac OS, care fusese introdus pe piață mai înainte de către compania Apple Computers, astăzi numită Apple Inc.

Marea majoritate a calculatoarelor utilizate în activitățile economice, sociale, de învățământ şi de cercetare folosesc una dintre versiunile sistemului de operare Windows. Dacă în 2004 Windows deținea aproximativ 90 % din piața de sisteme de operare, acum se estimează că aproximativ 85 % din calculatoarele de tip *Personal Computer* utilizează sisteme windows.

Din acest motiv, cunoaşterea principiilor şi mecanismelor pe care se sprijină sistemele Windows, constituie un element esențial al formării specialiştilor informaticieni.

M6.U2.2. Obiectivele unității de învățare

Această unitate de învățare îşi propune ca obiectiv principal o inițiere a studenților în conceptele generale asupra sistemelor Windows. La sfârşitul acestei unități de învățare studenții vor fi capabili să:

- înțeleagă şi să explice evoluția sistemelor Windows;
- înțeleagă şi să explice organizarea sistemelor Windows;
- înțeleagă şi să explice structura nucleului Windows;
- înțeleagă şi să explice interfațele oferite de sistemul Windows;
- înțeleagă şi să explice organizarea sistemului de fişiere sub Windows;
- înțeleagă şi să explice administarea proceselor sub Windows;
- înțeleagă şi să explice gestiunea memoriei sub Windows.

Durata medie de parcurgere a unității de învățare este de 3 ore.

M6.U2.3. Generalităţi despre sistemele Windows

Evoluţia sistemelor Windows. Primele versiuni de Windows (1.0- lansată în 1985, 2.0- lansată în 1987, 3.0- lansată în 1990 şi cele care au urmat-o 3.1 şi 3.11), au fost lansate de Microsoft ca interfeţe grafice(GUI), care se utilizau peste sistemul de operare MS-DOS.

În 1993, apare Windows NT, prima versiune de sistem de operare Windows, care este un sistem de operare pentru calculatoare pe 32 de biţi, care suportă aplicaţiile scrise DOS şi are aceeaşi interfaţă cu Windows 3.1.

1995, sunt lansate versiunile Windows NT 4.0 şi Windows 95. Windows 95 a reprezentat o variantă îmbunătăţită a Windows 3.1. Windows NT 4.0 are aceeasi arhitectura interna ca şi versiunile Windows 3.x şi furnizează aceeaşi interfaţă utilizator ca şi Windows 95.

Schimbarea arhitecturală majoră este că mai multe componente grafice care se executau în mod utilizator, ca şi parte a subsistemului Win32 (în versiunile 3.x), au fost mutate în executivul Windows NT, care se execută în modul nucleu, ceea ce are ca avantaj creşterea vitezei de operare al acestor functii importante. Un potenţial pericol este că aceste funcţii grafice se execută accesând serviciile de nivel scăzut ale sistemului, ceea ce poate afecta fiabilitatea sistemului de operare.

Windows 98 a fost succesorul lui Windows 95. Versiunea 5.0 a NT a fost lansată în anul 2000 şi fost redenumită Windows 2000(Win2K). Architecturile executivului şi nucleului sunt cam aceleaşi cu cele ale NT 4.0, dar au fost adăugate câteva componente importante. În Windows 2000 s-au adăugat servicii şi funcţii ca suport al prelucrării distribuite. Componenta esentială introdusă în acest scop este Active Directory; acesta este un serviciu de directoare distribuit (colecţie de informaţii despre obiecte care sunt în legătură unele cu altele într-o anumită privinţă), utilizat pentru: a organiza şi simplifica accesul la resursele unei reţele de calculatoare; aplicarea securităţii pentru a proteja obiectele din cadrul reţelei faţă de intruşii exteriori sau utilizatorii interni; distribuirea resurselor directorului la calculatoarele din reţea; replicarea directorului pentru a o face disponibilă la mai mulţi utilizatori; separarea directorului în mai multe bucăţi care sunt stocate pe diferite calculatoare din reţea.

De asemenea, Windows 2000 a îmbunătaţit facilităţile de “plug-and-play” şi “power-management” (importanta pentru laptop-uri). Sub Windows 2000 se face distincţie între Windows 2000 Server şi Windows 2000 desktop. În esenţă, arhitecturile şi serviciile nucleului şi executivului sunt aceleaşi, dar Server include anumite servicii necesare utilizării serverelor de retea, cum sunt cele de securitate.

Au existat patru versiuni ale sistemului Windows 2000: versiunea Profesional este destinată calculatoarelor individuale; celelalte trei versiuni(Server, Advanced Server şi DataCenter Server) sunt destinate calculatoarelor server dintr-o reţea. Windows 2000 DataCenter Server este destinat serverelor multi-procesor şi poate fi instalat pe sisteme care au până la 32 de procesoare şi până la 64 G de memorie.

Windows 95/98 şi Windows ME(Milenium Edition) formează sub-familia sistemelor Windows 9x, care este destinată calculatoarelor individuale, neconectate în reţea. Windows ME este o variantă îmbunătăţită a lui Windows 98, care la rândul lui perfecţionează funcţiile lui Windows 95. Windows 9x/ME diferă de Windows NT/2k prin faptul că implementează mai puţine funcţii Win32API. Windows NT/2K conţine un model de securitate care nu este întâlnit la sistemele Windows 9x/ME, deoarece Windows NT/2K este destinat calculatoarelor care sunt conectate în reţea. Facilităţile de conectare conţinute de Windows 9x/ME sunt o sub-mulţime a celor conţinute de Windows NT/2K. O altă deosebire constă în faptul că Windows NT permite aplicaţiilor să manipuleze anumiţi parametri, care influenţează comportarea administratorului memoriei virtuale.

Windows CE(Consumer Electronics) este destinat calculatoarelor personale care utilizează anumite echipamente electronice casnice, cum ar fi televizoare; ele pot fi legate la rețelele de televiziune prin cablu. Setul său de funcții Win32API este cel mai redus, fiind orientat mai ales către jocuri.

În 2001 a fost lansată o nouă distribuție desktop - Windows XP, cu versiuni pe calculatoare pe 32 sau 64 de biți. Fiind succesorul lui Windows 2000, el poate fi utilizat ca sistem de operare pentru calculatoarele legate în rețea, dar poate înlocui şi sistemele Windows 95/98/ME destinate calculatoarelor individuale. Windows XP propune o interfață grafică nouă, care se bazează pe evoluțiile componentelor hardware şi urmăreşte utilizarea facilă a interfeței sistemului de către utilizatori. Modelul de securitate adoptat este superior tuturor versiunilor de Windows anterioare. De asemenea, sistemul oferă facilități superioare de lucru în rețea.
Versiunea XP propune două variante:
- Windows XP Net Server este utilizat de calculatoarele server dintr-o rețea; această variantă oferă o serie de facilități şi mecanisme noi, printre care: lansarea simultană de mai multe sesiuni de pe calculatoarele din rețea, posibilitatea ca utilizatorul să poată comuta rapid printre serviciile oferite de sistem.
- Windows XP Proffesional este destinat calculatoarelor individuale precum şi calculatoarelor client legate în rețea.
În 2003, a apărut o nouă versiune server - Windows Server 2003, care suportă procesoare atât pe 32 de biți, cât şi pe 64 de biți. În 2007, a fost lansat Windows Vista, care suporta arhitecturile procesoarelor existente. S-au adus îmbunatatiri ale modelului de securitate. Windows Server 2008 este varianta coresp. care se instaleaza pe servere.

În 2007 apare Windows Vista. **Securitatea** a reprezentat obiectivul principal pentru acest nou sistem de operare. Astfel, au fost incluse noi facilități de securitate care fac mai dificilă expunerea sistemului de operare la atacurile informatice. Acestea avertizeaza utilizatorul asupra oricărei activități suspecte şi cer confirmarea înainte de acordarea accesului la sistem. Sistemul conține o aplicație care va proteja computerul fată de amenințările informatice care pot determina publicarea de informații confidențiale în exterior, cu sau fără voia utilizatorului; va trebui procurată separat o aplicație antivirus, pentru că noul sistem de operare nu conține o protecție şi la aceste atacuri.

Exemplu. „One Care", care este bazată pe tehnologie romanească, urmare a achiziției companiei româneşti Gecad de către Microsoft în 2003.

Protecția stabilității sistemului este îmbunătățita, astfel încât noul sistem de operare are disponibile facilități ce permit îmbunatățirea continuității operării prin restaurarea stării sistemului la un punct anterior în timp, astfel încât dupa instalarea software-ului pentru o componentă hardware sau a unei aplicații care duce la blocarea sau instabilitatea sistemului se poate reveni la starea inițială. Protectia informațiilor stocate a făcut un pas înainte, din două puncte de vedere: Vista este capabil să recreeze varianta anterioară pentru anumite fişiere şi directoare; software-ul pentru realizarea de copii de siguranță a informațiilor stocate a fost îmbunătățit, putând realiza salvări periodice mai uăor ăi cu mai multe facilități.

Din punctul de vedere al **performanțelor**, testele efectuate au arătat că Windows Vista rulează la fel sau mai lent comparativ cu Windows XP. Noile opțiuni adăugate din punct de vedere al functionalității sau al securității necesită mai multă putere de calcul şi memorie mai multă.

În ceea ce priveşte **comunicația** lucrurile au fost simplificate şi îmbunătățite. Este mult mai simplu să conectezi un computer la rețea, locală sau Internet, indiferent de tipul conexiunii,

prin cablu sau fără fir şi este mult mai sigur, Microsoft adăugând în Vista standarde mult mai sigure pentru transferurile de informaţie.

Această distribuţie a introdus **interfaţa grafica 3D**. Aceasta foloseşte la maximum noile plăci video şi oferă o nouă imagine asupra tradiţionalului desktop, în 3 dimensiuni. Au fost introduse a**plicaţii noi**.

Exemplu. „Sidebar", care, instalată pe desktop, furnizează informaţii despre ora exactă, vreme sau poate afişa ultimele ştiri; „Gallery" este o aplicaţie de afişare şi gestionare a fotografiilor digitale.

Printre îmbunătăţirile aduse de Microsoft în viitoarea versiune Windows 7 (lannsat în 2009) se numără:
- un taskbar îmbunătăţit;
- noi moduri de manipulare şi organizare a ferestrelor;
- un nou Internet Explorer, ajuns acum la versiunea 8 ;
- o integrare mai bună cu serviciile online Windows Live;
- un management al perifericelor şi al dispozitivelor conectabile la PC mult mai bine pus la punct;
- o aplicaţie “vrăjitor” de conectare/creare a unei reţele mult mai intuitiv;
- o performantă mai bună comparativ cu Windows Vista;
- opţiuni noi de optimizare a consumului bateriei la laptopuri;
- mai multe opţiuni de personalizare;
- functii noi pentru sistemele dotate cu ecrane sensibile la atingere şi capabilităţi de recunoaştere a scrisului.

Caracteristici generale ale sistemelor Windows. Windows înlocuieşte modul de lucru text cu cel grafic şi introduce, conceptual următoarele elemente:

- permite afişarea într-o mulţime de ferestre organizate ierarhic;
- textele pot fi afişate într-o mare gamă de fonturi scalabile;
- permite interceptarea de evenimente; un eveniment este interceptat atunci când se apasă o anumită tastă sau un buton de la mouse (evenimentul este interceptat diferit, în funcţie de poziţia cursorului în momentul evenimentului);
- oferă o serie de mecanisme de captare a unor informaţii: ”butoane” cu răspuns unic sau “ferestre ” de alegere a unuia sau mai multor răspunsuri posibile, dintr-o listă predefinită;
- oferă posibilitatea de comunicare între programele utilizator şi mediul Windows; în legătură cu această ultimă posibilitate, se introduce o nouă fază în dezvoltarea de programe şi anume adăugarea de resurse Windows în aplicaţii. Astfel, după link-editarea programului, acesta este completat cu o serie de module, care permit programului să manevreze cu elemente Windows: ferestre, butoane, evenimente etc.

Execuţia simultană a mai multor lucrări. Sub Windows multi-tasking-ul este oarecum “aproximativ “, în sensul următor: O aplicaţie lansată poate fi suspendată temporar. După suspendare, poate fi lansată o alta sau poate fi relansată una dintre cele suspendate. Singurele aplicaţii efectiv executabile în paralel sunt cele oferite de DOS, cum ar fi de exemplu listarea la imprimantă în paralel cu execuţia altor programe şi supravegherea ceasului sistem în vederea lansării în execuţie a unei lucrări la o oră fixată.

Odată cu versiunile Windows 95/98, a apărut un mod mai uşor de a comuta între programele lansate, şi anume bara de aplicaţii. Când lansaţi un program, Windows adaugă un buton pentru acel program la bara de aplicaţii. Atunci când lansaţi mai multe programe, Windows rearanjează automat butoanele în aşa fel încât să puteţi vedea tot timpul acele butoane. Butonul asociat programului activ are o luminozitate deosebită faţă de celelalte programe de pe bara de aplicaţii.

Schimbul de date între aplicații. Windows gestionează o aplicație numită ClipBoard. Prin intermediul ei, utilizatorul poate depozita temporar, din orice aplicație, informații în zona rezervată ClipBoard, care pot fi utilizate de către oricare altă aplicație.

Interfața grafică(GUI-Grafics User Interface). Comunicarea între utilizator şi sistem se poate face prin intermediul tastaturii sau a mouse-ului. Prin intermediul tastaturii, se poate selecta o tastă sau o combinație de taste, corespunzătoare unei anumite acțiuni iar prin intermediul mouse-ului se pot selecta unul sau mai multe „butone", din anumite entități grafice, pe care le vom descrie pe scurt în cele ce urmează. Elementul esențial de afişare în această interfață este fereastra, care reprezintă o porțiune dreptunghiulară afişată într-o zonă a ecranului, destinată comunicării dintre utilizator şi sistem. Sub Windows, orice fereastră conține câteva elemente comune, şi anume:

►**Bara de titlu** (Caption) din partea cea mai de sus a ferestrei afişează numele aplicației.

►Imediat sub bara de titlu este **bara de meniuri** (Menu Bar), care afişează meniurile disponibile (opțiunile specifice aplicației care rulează în momentul respectiv).

►**Bara de unelte (Toolbar)** conține butoane şi alte elemente care vă permite să lansați comenzi. Conținutul barei de unelte variază de la fereastră la fereastră, depinzând de fereastră sau aplicație, dar poziția ei este este constantă.

►În fereastra dreptunghiulară se află icon-uri, mici imagini sau pictograme, fiecare reprezentând părți din sistemul de calcul şi aplicațiile care controlează sistemului.

►**Butoanele de minimzare (Minimize), de maximizare (Maximize) şi de închidere (Close)** apar în colțul din dreapta-sus al ferestrei pe aceeaşi linie ca şi bara de titlu. Butonul de minimizare reduce fereastra la un buton pe bara de aplicații, iar butonul de maximizare măreşte ferestra la dimensiunea întregului ecran. Butonul de închidere reprezintă o modalitate rapidă de a închide fereastra. După folosirea butonului de maximizare, acesta se transformă în butonul de restaurare (Restore button). Butonul de restaurare, permite aducerea ferestrei la mărimea ei anterioară (mai mică).

►**Marginile (borders)** sunt cele patru linii care definesc limitele unei ferestre**.**

►**Bara de informații (status bar)** furnizează informații. Pe măsură ce se aleg opțiuni din meniu, se selectează obiecte din ferestre sau se lansează comenzi, acțiunea este descrisă pe bara de informații.

►**Tab-ul de redimensionare** (resize tab) oferă o suprafață mare care se poate „agăța" cu mouse-ul atunci când se doreşte modificarea mărimii unei ferestre.

►Atunci când o fereastră nu este suficient de mare pentru a afişa tot conținutul ei, apare butonul de derulare.Trăgând de **glisor** (butonul pătrat de pe bara de derulare) se mută conținutul ascuns al ferestrei în partea vizibilă a acesteia. De asemenea, se pot folosi săgețile de la capetele barelor de derulare pentru a obține acelaşi rezultat.

► **Desktop(Biroul electronic Windows).** După instalarea completă a sistemului apare pe ecran imaginea similară unui birou. Acesta poate conține un ceas, un calculator de birou, un loc de depozitare a fişierelor şi dosarelor, o listă completă a tuturor instrumentelor şi dosarelor etc. **Bara de aplicații** conține butonul **Start**, care reprezintă cheia către instrumentele, accesoriile şi caracteristicile Windows. Când se apasă butonul Start, Windows afişează o listă de opțiuni (numită pop-up) din care se poate face o alegere. Pentru fiecare aplicație lansată, pe bara de aplicații apare un buton cu numele aplicației.

Organizarea sistemelor Windows. Structura sistemului este descrisă în figura 6.2.1. Observăm organizarea modulară şi pe niveluri a sistemului. Principalele niveluri sunt cel de **abstractizare a hardwarelui** (HAL), **nucleul** şi **executivul**. Toate aceste componente sunt executate în mod protejat. Subsistemele se împart în două categorii: **subsistemele de mediu**, care emulează diferite sisteme de operare şi **subsistemul de protecție(securitate)**.

Executivul Windows oferă o mulțime de servicii, pe care le poate folosi oricare subsistem de mediu: administrarea obiectelor, administrarea memoriei, administrarea proceselor, administrarea I/O, lucrul cu apeluri de proceduri locale, monitorizarea securității, mecanismul „plug-and-play” şi autoîncărcarea sistemului.

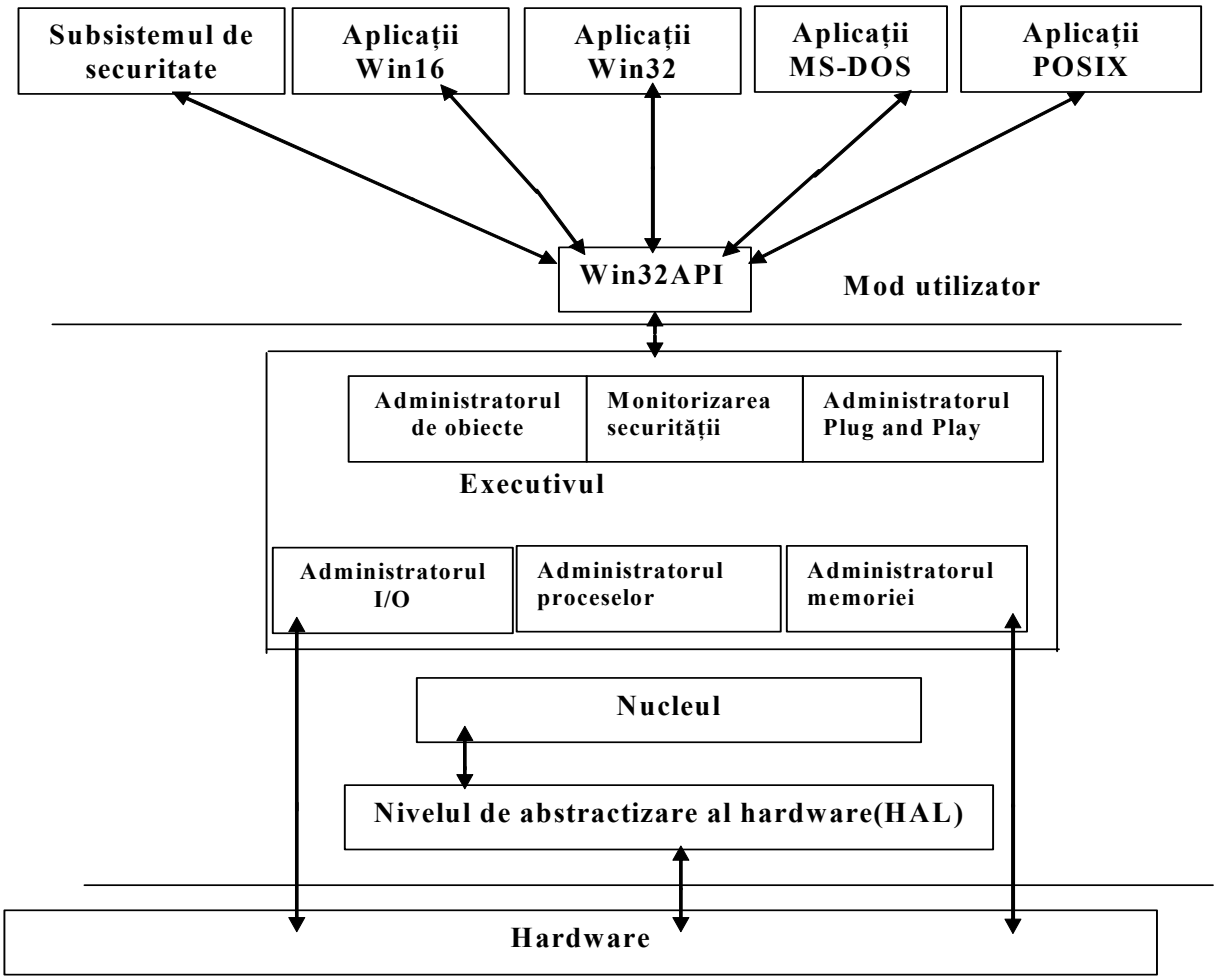


Figura 6.2.1. Structura sistemului Windows

Interfața sistemului cu utilizatorul. Fiecare dintre sistemele de operare Windows importă o submulțime a unei interfețe unice, denumită Win32API. Această API (**A**pplication **P**rogramming **I**nterface) este cuprinzătoare şi dinamică. Diferitele versiuni Windows implementează submulțimi ale Win32API. Rațiunea pentru care a fost creată o singură API, este leagată de necesitatea portabilității între diferite sisteme Windows. Mulțimea de funcții implementate de către Windows 95/98/ME include pe cele utilizate de către Windows CE. Windows NT, Windows 2000 şi Windows XP implementează o mulțime de funcții care le include pe cele implementate de către Windows 95/98/ME(vezi figura urm.). Nucleul lui Win2K şi subsistemele Win32 implementează toate funcțiile Win32API, fiind cel mai complex dintre membrii familiei apărute până la acea dată. Win2K foloseşte un model de calcul bazat pe procese şi fire de execuție şi se sprijină pe conceptele proiectării şi programării orientate pe obiecte.

Să ne reamintim...

Versiunile Windows 1.0, 2.0, 3.0, 3.1 au fost lansate de Microsoft ca interfețe grafice(GUI), care se utilizau peste sistemul de operare MS-DOS. Windows NT este prima versiune de sistem de operare Windows, care este un sistem de operare pentru calculatoare pe 32 de biți, care suportă aplicațiile scrise DOS şi are aceeaşi interfață cu Windows 3.1. Windows 95 a reprezentat o variantă îmbunătățită a Windows 3.1. Windows NT 4.0 are aceeasi arhitectura interna ca şi versiunile Windows 3.x şi furnizează aceeaşi interfață utilizator ca şi Windows 95. Windows 98 a fost succesorul lui Windows 95. Versiunea 5.0 a NT a fost lansată în anul 2000 şi fost redenumită Windows 2000(Win2K). Windows XP este succesorul lui Windows 2000; el poate fi utilizat ca sistem de operare pentru calculatoarele legate în rețea, dar poate înlocui şi sistemele Windows 95/98/ME destinate calculatoarelor individuale. În 2007 apare Windows Vista, oal cărui biectiv principal a fost securitatea. Windows 7este varianta îmbunătățită a Windows Vista.

Windows înlocuieşte modul de lucru text cu cel grafic şi permite execuția simultană a mai multor lucrări.

Sistemele Windows sunt organizate pe niveluri. Principalele niveluri sunt cel de abstractizare a hardwarelui (HAL), nucleul şi executivul.

Înlocuiți zona punctată cu termenii corespunzători.

1. Schimbarea arhitecturală majoră Windows NT este că mai multe ..... care se executau în mod ....., au fost mutate în .....
2. În Windows 2000 s-au adăugat servicii şi funcții ca suport al ..... Componenta esențială introdusă în acest scop este ……
3. Versiunea XP propune varianta.....destinată calculatoarelor ..... dintr-o rețea şi ..... destinată calculatoarelor ..... precum şi calculatoarelor ..... legate în rețea.
4. În Windows Vista ..... a reprezentat obiectivul principal pentru acest nou sistem de operare.
5. Îmbunătățirile aduse de Microsoft în viitoarea versiune Windows 7 sunt:.....

M6.U2.4. Componente de nivel scăzut ale sistemelor Windows

Nivelul de abstractizare a hardware. Unul dintre obiectivele Windows a fost portabilitatea (sistemul sa funcționeze pe o clasă cît mai mare de calculatoare, indiferent de producător). Portabilitatea este asigurată de faptul că majoritatea componentelor sistemului sunt scrise în C sau C++. De asemenea, tot codul care este dependent de un anumit procesor, se află în fişierele DLL(Dynamic Link Library), care se găsesc în componenta de abstractizare a hardware (HAL) a sistemului.

Un fişier DLL realizează maparea spațiului de adrese al procesului, astfel încât orice orice funcție implementată de către DLL apare ca o componentă a procesului. HAL manipulează direct componenta hardware a sistemului, astfel că restul sistemului Windows funcționează independent de aceasta.

HAL reprezintă cel mai scăzut nivel al sistemului de operare, ce ascunde diferențele hardware dintre diferitele clase de sisteme de calcul, ceea ce face din Windows un sistem de operare portabil. HAL exportă o interfață virtuală care este folosită de către nucleu, executiv şi driverele de unitate. Un avantaj al acestei abordări, este că este necesară numai o versiune a

fiecărui driver, care poate fi executată indiferent de platforma hardware, deci driverele nu trebuie să mai fie adaptate de la un sistem la altul. Din motive de performanță, driverele de I/O şi cele grafice pot accesa direct componenta hardware.

Nivelul nucleu. Deasupra HAL se află nucleul şi driverele de dispozitiv; el stă la baza executivului şi a subsistenelor. În concepția Microsoft, obiectivul nucleului este să facă restul sistemului de operare complet independent de hard (portabil). Nucleul continuă de acolo de unde se opreşte HAL. Nucleul accesează hard-ul prin intermediul HAL şi realizează abstractizări de nivel mai înalt decât HAL.

Exemplu. HAL conține apeluri pentru asocierea procedurilor de tratare a întreruperilor cu întreruperile. În schimb, nucleul realizează schimbarea de context.

Nucleul realizează planificarea firelor de execuție; atunci când este momentul pentru a verifica dacă un fir de execuție nou se poate executa, nucleul alege firul de execuție şi face schimbarea de context necesară rulării lui.

Exemplu. Situații când nucleul verifică dacă un fir de execuție nou se poate executa sunt: după ce a expirat o cuantă de timp sau după terminarea unei întreruperi de I/O.

Nucleul sistemului este orientat pe obiecte. El execută o anumită sarcină, folosind o mulțime de obiecte ale sale, ale căror atribute conțin date şi ale căror metode execută activități specifice lui. Nucleul asigură suport pentru utilizarea a două clase de obiecte de nivel scazut: Obiecte de control şi Obiecte dispecer.

Obiectele de control sunt acele obiecte care controleaza sistemul. Acestea sunt:
- apeluri de proceduri întârziate (DPC-Deffered Procedure Call); un obiect DPC este folosit pentru a separa partea care nu este critică de partea critică, din punct de vedere al timpului dintr-o procedură de tratare a întreruperii. În general, o procedură de tratare a întreruperii salvează câțiva regiştri hard volatili asociați cu dispozitivele de I/O care au generat întreruperea, astfel încât aceştia să nu fie şterşi şi reactivează echipamentul, dar lasă majoritatea prelucrărilor pentru mai târziu.

Exemplu. După ce a fost apăsată o tastă, procedura de tratare a întreruperilor tastaturii citeşte codul tastei dintr-un registru şi reactivează întreruperea tastaturii, dar nu are voie să prelucreze tasta imediat, dacă o altă activitate, cu o prioritate mai înaltă trebuie executată.

De asemenea, DPC-urile sunt folositoare pentru expirarea contoarelor şi alte activități a căror procesare efectivă nu trebuie să fie instantenee. Coada DPC este mecanismul prin care se memorează ceea ce trebuie făcut mai târziu.
- apeluri asincrone de procedură(APC); sunt asemănătoare DPC-urilor, cu excepția faptului că ele se execută în contextul unui proces specific.

Exemplu Atunci când se prelucrează apăsarea unei taste, nu contează în ce context va rula DPC-ul deoarece singurul lucru care se va intimpla este că codul tastei va fi pus îintr-un tampon. În schimb, dacă întreruperea necesită copierea unui tampon din zona nucleu în zona utilizator, atunci procedura de copiere trebuie să ruleze în contextul unui proces destinatar. Contextul destinatar este necesar pentru ca tabela de pagini să conțină ambele tampoane.
- obiectele întrerupere, asociază sursa întreruperii cu o anumită clasă de întreruperi;

- obiecte de urmărire a tensiunii de alimentare, pentru a se apela automat o anumită rutină când apare o cădere de tensiune;
- obiectul proces reprezintă spațiul de adrese virtuale şi informațiile de control necesare execuției firelor de control ale procesului respectiv;
- obiectul profil este utilizat pentru a măsura cantitatea de timp necesar unui bloc de cod.

Obiecte ale dispecerului sunt:
- obiectele eveniment sunt folosite pentru a înregistra apariția evenimentelor şi pentru a se realiza sincronizarea lor cu anumite acțiuni;
- obiectele de comutare sunt utilizate pentru comutatea sistemului din modul nucleu(protejat) în cel utilizator şi invers;
- obiectele semafor sunt folosite ca porți de intrare pentru a controla numărul firelor de execuție care accesează anumite resurse; cele cu excludere mutuală(mutex) sunt utilizate numai în modul nucleu, pentru a se rezolva problema interblocării;
- obiectele „thread" sunt entități care sunt executate de către nucleu, care aparțin unor clase proces;
- obictele ceas sunt folosite pentru măsurarea intervalelor de timp şi pentru a semnala depăşirile de timp(timeout), când timpul de efectuare a operațiilor este mai lung decât cel aşteptat şi acestea trebuie să fie întrerupte.

Manipularea întreruperilor. Windows defineşte mai multe situații de excepție software şi hardware, independente de arhitectura sistemului de calcul, cum ar fi: violarea drepturilor de acces la memorie, împărțire la zero, instrucțiune ilegală, depăşire a domeniului de valori pentru o expresie, eroare la citirea unei pagini etc. Aceste situații conduc la abandonarea fluxului de control pe care procesorul îl urmează în mod normal şi sunt rezolvate prin intermediul întreruperilor.

Manipularea excepțiilor se poate realiza în mod nucleu sau utilizator. Dispecerul situațiilor de excepție creează o înregistrare care conține motivul acelei excepții şi caută manipulatorul excepției respective, care poate prelucra acea înregistrare.

Tipul şi numărul întreruperilor depinde de tipul procesorului. Din motive de portabilitate, dispecerul de întreruperi realizează o corespondență între întreruperile hardware şi o mulțime standard. Fiecare întrerupere are un număr de prioritate, în funcție de care se realizează servirea. Nucleul foloseşte o tabelă prin care se asociază la fiecare nivel de întrerupere o anumită rutină de serviciu.

Exemplu Sub Windows XP există există 32 nivele de întreruperi(IRQL-Interrupt ReQuest Level); opt dintre ele sunt rezervate nucleului, iar celelalte 24 reprezintă întreruperi hardware care se realizeză prin intermediul HAL.

Întreruperi DPC. O procedură DPC (Defered Procedure Call) îndeplineşte funcții de sistem şi este executată în mod supervizor. Atributul „amânat" (deffered) se referă la faptul că execuția ei este amânată, până când IRQL-ul ei coboară la un nivel destul de scăzut. Acest tip de procedură este utilizat de către nucleu, pentru procesarea evenimentelor de expirare a ceasurilor de timp real şi de către drivere în prelucrarea cererilor de I/O. DPC-urile sunt puse în coadă la nivelul 2, de către rutine care se execută la niveluri de prioritate superioare. Odată coborât IRQL, DPC-urile din coadă vor fi lansate în execuție, folosind resursele firului care se află în momentul curent în execuție, fără ca acesta să fie informat în vreun fel.

Sincronizarea proceselor la nivel scăzut se realizează prin intermediul procedurilor APC(Asynchronous Procedure Call), care sunt executate în contextul unui anumit fir. Aceste proceduri sunt păstrate în liste asociate firelor de execuție, spre deosebire de procedurile DPC care sunt păstrate într-o listă unică de sistem şi sunt lansate atunci când firul corespunzător se

află în execuție. Procedurile APC sunt destinate situațiilor care necesită operare într-un anumit spațiu de adrese.

Exemplu Copierea datelor transferate de către un fir care a inițiat o cerere de I/O în/din bufferul unui controller.

Refacerea după o cădere de tensiune. Întreruperea datorată unei pene de curent atenționează sistemul de operare, atunci când se petrece un astfel de eveniment. Obiectul folosit pentru notificarea căderii de tensiune furnizează informațiile necesare driverului de unitate de a apela o rutină care este lansată în momentul în care există din nou tensiune de alimentare şi care va „reinițializa" unitatea respectivă.

Să ne reamintim...
HAL reprezintă cel mai scăzut nivel al sistemului de operare, ce ascunde diferențele hardware dintre diferitele clase de sisteme de calcul, ceea ce face din Windows un sistem de operare portabil. HAL exportă o interfață virtuală care este folosită de către nucleu, executiv şi driverele de unitate.

Un fişier DLL realizează maparea spațiului de adrese al procesului, astfel încât orice orice funcție implementată de către DLL apare ca o componentă a procesului. HAL manipulează direct componenta hardware a sistemului, astfel că restul sistemului Windows funcționează independent de aceasta.

Deasupra HAL se află nucleul şi driverele de dispozitiv; el stă la baza executivului şi a subsistenelor. În concepția Microsoft, obiectivul nucleului este să facă restul sistemului de operare complet independent de hard (portabil).

Windows defineşte mai multe situații de excepție software şi hardware, independente de arhitectura sistemului de calcul. Manipularea excepțiilor se poate realiza în mod nucleu sau utilizator. Dispecerul situațiilor de excepție creează o înregistrare care conține motivul acelei excepții şi caută manipulatorul excepției respective, care poate prelucra acea înregistrare.

Înlocuiți zona punctată cu termenii corespunzători.

1. Tot codul care este dependent de un anumit procesor, se află în ....., care se găsesc în ..... a sistemului.
2. Un fişier DLL realizează ....., astfel încât orice orice funcție implementată de către DLL apare ca o componentă a .....
3. HAL reprezintă ..... al sistemului de operare, ce ascunde ..... hardware dintre diferitele clase de sisteme de calcul, ceea ce face din Windows un sistem de operare .....
4. Nucleul accesează hard-ul prin ..... şi realizează ..... de nivel mai înalt decât .....
5. Un obiect DPC este folosit pentru a separa ....., din punct de vedere al .....dintr-o procedură de tratare a .....
6. DPC-urile sunt folositoare pentru ..... şi alte activități a căror procesare ..... nu trebuie să fie .....
7. O procedură DPC îndeplineşte funcții ..... şi este executată în ..... Atributul „amânat" se referă la faptul că ..... ei este amânată, până când ..... coboară la un nivel destul de scăzut.

M6.U2.5. Administrarea obiectelor

La nivel de executiv sunt utilizate o serie de clase de obiecte (director, semafor, eveniment, proces sau thread, port, fişier). Atunci când un obiect este creat, managerul de obiecte alocă un bloc de memorie virtuală din spaţiul de adrese al nucleului; reciproc, când un obiect este dealocat, spatiul respectiv este eliberat.

Rolul administratorului de obiecte este de a superviza utilizarea obiectelor. Când un thread doreşte să folosească un obiect, el apelează metoda **open,** pentru a obţine accesul la manipulatorul acelui obiect, care este o interfaţă standard către toate tipurile de obiecte. Deoarece administratorul de obiecte este singura entitate care poate genera un manipulator de obiect, el face parte din modulul de securitate al sistemului.

Exemplu. Administratorul obiectelor verifică dacă un proces are drepturile de acces la un anumit obiect, în momentul în care încearcă să deschidă acel obiect.

Administratorul obiectelor poate urmări care dintre procese foloseşte un anumit obiect. Fiecare antet(header) de obiect, conţine un contor al numărului de procese care au accesat obiectul respectiv. Acest contor este incrementat, respectiv decrementat, când un proces cere, respectiv nu mai utilizează acel obiect. Dacă obiectul respectiv este temporar, când valoarea contorului a ajuns la 0, obiectul respectiv este şters din spaţiul de nume.

Orice obiect are un nume; spaţiul de nume al proceselor are un caracter global, deci obiectele create de un proces pot fi partajate împreună cu alte procese. Numele unui obiect poate fi permanent sau temporar. Un nume permanent reprezintă o entitate cu caracter permanent, chiar dacă nu este utilizată la un moment dat, cum este cazul unui driver de disc. Numele temporare există numai atâta timp cât anumite procese păstrează manipulatoare către aceste obiecte.

Numele de obiecte sunt structurate la fel ca o structură de directoare arborescentă. Directoarelor le corespund obiectele director, care conţin numele tuturor obiectelor din acel director; în plus, acestea pot conţine domenii de obiecte care sunt mulţimi de obiecte. Astfel, există posibilitatea construcţiei ierarhice a unei structuri de obiecte.

Exemplu. Un domeniu de obiecte poate fi format din driverele de discuri. În momentul când un disc este introdus în sistem, spaţiul de nume al acestuia este adăugat celui deja existent.

Administratorul de obiecte crează manipulatori de obiecte, care constă din informaţii de control al accesului şi un pointer la obiect.

Un proces poate obţine un manipulator de obiecte prin crearea obiectului de către el însuşi, primirea unui duplicat al unui manipulator de la un alt obiect sau prin moştenirea unui manipulator de la un proces părinte. Tabela de obiecte a procesului conţine toate aceste manipulatoare. O intrare în tabela de obiecte conţine drepturile de acces şi stabileşte dacă manipulatorul poate fi moştenit de către procesele fiu. Când un proces se termină, sistemul închide toate manipulatoarele de obiecte deschise de către acesta.

Când un utilizator se loghinează la un sistem şi trece de faza de autentificare, îi este ataşat un obiect care conţine o serie de atribute, printre care: identificatorul de securitate, grupurile din care face parte, grupul primar, lista control a accesului. Aceste atribute determină care dintre serviciile şi obiectele sistemului pot fi folosite de către un utilizator dat.

Fiecare obiect este protejat de către o listă de control a accesului, care conține identificatorii de securitate şi drepturile acordate fiecărui proces. Când un proces încearcă să deschidă un obiect, sistemul compară identificatorul lui de securitate, corespunzător utilizatorului căruia îi aparține procesul respectiv, cu lista de control a accesului la obiect, pentru a determina dacă accesul este permis.

Să ne reamintim...

La nivel de executiv sunt utilizate o serie de clase de obiecte (director, semafor, eveniment, proces sau thread, port, fişier). Atunci când un obiect este creat, managerul de obiecte alocă un bloc de memorie virtuală din spațiul de adrese al nucleului; reciproc, când un obiect este dealocat, spatiul respectiv este eliberat.

Numele de obiecte sunt structurate la fel ca o structură de directoare arborescentă. Directoarelor le corespund obiectele director, care conțin numele tuturor obiectelor din acel director; în plus, acestea pot conține domenii de obiecte care sunt mulțimi de obiecte. Astfel, există posibilitatea construcției ierarhice a unei structuri de obiecte.

Înlocuiți zona punctată cu termenii corespunzători.

1. Când un thread doreşte să folosească un obiect, el apelează metoda ....., pentru a obține accesul la ..... acelui obiect, care este o ..... către toate tipurile de .....
2. Fiecare ..... de obiect, conține un ..... al numărului de ..... care au accesat obiectul respectiv.
3. Administratorul de obiecte crează .....de obiecte, care constă din informații de ..... al ..... şi un ..... la obiect.
4. Când un utilizator se loghinează la un sistem şi trece de faza de autentificare, îi este ataşat un obiect care conține o serie de atribute, printre care: ..... Aceste atribute determină care dintre ..... şi ..... sistemului pot fi folosite de către un ..... dat.
5. Fiecare obiect este protejat de către ....., care conține identificatorii de ..... şi .....acordate fiecărui proces.

M6.U2.5. Memoria virtuală sub Windows

Vom discuta gestionarea memoriei pentru sistemele de calcul cu procesor pe 32 de biți; în cazul celor pe 64 de biți, modalitatea de administrare este asemănătoare. Fiecărui proces sub Windows, îi este acordat un spațiu de adrese virtuale de dimensiune fixă (4 Go), care este mult mai mare decât dimensiunea memoriei primare a unui calculator. Procesul nu foloseşte tot spațiul de adrese virtuale, ci numai ceea ce îi este necesar, de obicei mult mai puțin din ceea ce îi este alocat.

O parte a spațiului de memorie virtuală (de obicei 2 Go), este folosit pentru a permite firului de execuție referențierea unor obiecte din spațiul de memorie primară, iar restul este folosit pentru a referenția adrese folosite de către sistemul de operare (spațiu utilizat în mod supervizor).

Dimensiunile spațiilor de adrese virtuale utilizate de proces, respectiv de nucleul SO diferă de la o variantă la alta de sistem Windows. Chiar dacă partea de adrese supervizor există în spațiul de adrese al procesului, memoria poate fi referită de un fir de execuție al procesului, numai dacă acesta se găseşte în mod supervizor.

Sistemul de operare are nevoie de anumite mijloace pentru a determina cantitatea de spațiu de adrese necesare procesului. Editorul de legături construieşte imaginea statică a fişierului executabil, care defineşte spațiul de adrese virtuale. Anumite componente alocate dinamic(biblioteci ale sistemului), sunt adăugate spațiului de adrese virtuale în faza de execuție a procesului. Există două faze de adăugare dinamică de adrese la spațiul de adrese virtuale:

- rezervarea unei porțiuni a spațiului de adrese virtuale, numită regiune;
- cererea unui bloc de de pagini, dintr-o regiune a spațiului de adrese.

Fiecare procesor are o anumită unitate de alocare, adică dimensiunea minimă a blocului de adrese care va fi rezervat. De obicei, aceasta este de 64 Ko. De asemenea fiecare procesor are o anumită dimensiune a paginii, de obicei 4 sau 8 Ko. Memoria este cerută în bucăți formate din pagini, deci porțiunea cerută la un moment dat, este mult mai mică decât cea rezervată anterior.

Un fir de execuție din cadrul unui proces poate, în mod dinamic să rezerve o regiune a spațiului de adrese virtuale, fără a scrie nimic în fişierul de pagini din memoria secundară. De asemenea, el poate elibera o regiune a spațiului de adrese virtuale rezervată anterior. A doua fază este de a cere adrese, care anterior au fost rezervate, spațiu care este alocat în fişierul de pagini. Dacă firul de execuție al procesului, referențiază memoria alocată, pagina care conține adresa referențiată va fi încărcată din fişierul de pagini în memoria primară.

Translatarea adreselor se face cu ajutorul unor componente hardware, pentru a detecta rapid paginile de memorie virtuală ale procesului care lipsesc din memoria internă şi pentru a le pune în corespondență cât mai rapid pagini fizice. O adresă virtuală este memorată pe 32 de biți şi este generată de către procesor.

Win2K foloseşte o schemă de adresare structurată pe două niveluri. Cei mai puțin semnificativi K1 biți din cadrul adresei, reprezintă deplasamentul. Restul de biți se numeşte numărul de pagină virtuală şi este separat în două părți, denumite indexul tabelei de pagini (K2 biți) şi indicele directorului de pagini(K3 biți). Valorile lui K1, K2, K3 sunt specifice fiecărui procesor în parte. Valorile cele mai utilizate sunt de 12 biți pentru K1, respectiv de câte 10 biți pentru K2, K3.

Translatarea adreselor foloseşte aceste trei câmpuri astfel:
- Descriptorul de proces conține un pointer la începutul directorului de pagini, care are 1024 de intrări, pentru procesul dat. Indicele din director, este un deplasament unde este localizată o intrare în descriptorul de pagini (PDE- Page Descritor Entry). Fiecare proces poate avea mai multe tabele de pagini. PDE referențiază o anumită tabelă de pagini ce va fi folosită pentru această referință la memorie.
- Intrarea în tabelele de pagini (PTE – Page Table Entry), care la rândul lor au câte 1024 de intrări este găsită folosind indicele tabelei de pagini din adresă, ca un index în tabela de pagini. Dacă pagina căutată este încărcată în memoria internă în pagina fizică j, atunci PTE punctează către pagina fizică. Dacă ea nu este încărcată, administratorul memoriei virtuale trebuie să localizeze pagina în fişierul de pagini, să caute o pagină fizică, să o aloce procesului şi apoi să încarce pagina virtuală respectivă în pagina fizică.
- În final, deplasamentul este adăugat adresei paginii fizice, pentru a obține adresa baitului căutat din memoria primară.

Teoretic, directorul de pagini se poate memora în oricare dintre locațiile de memorie primară, dar de obicei el se află într-o locație fixă, specifică fiecărui procesor, a cărei adresă este salvată într-un registru. Conținutul acestui registru face parte din contextul firului de execuție

respectiv şi atunci când acesta nu mai beneficiază de serviciile CPU, conținutul registrului este salvat.

Windows foloseşte tabele de pagini multiple, pentru a face distincție între diverse utilizări ale spațiului de adrese. Cea mai evidentă diferențiere este că anumte pagini fac parte din spațiul utilizator, iar altele din spațiul supervizor.

Observație. Prin plasarea nucleului într-o tabelă de pagini separată, procese diferite au punct de intrare(PTE) către nucleu. Fiecare PTE referențiază un număr de pagină fizică, atunci când pagina corespunzătoare este încărcată. De asemenea, există o colecție de fanioane care descriu modul cum pagina poate fi referențiată (pagina respectivă este validă sau nu, este rezervată sau nu, dacă conținutul ei a fost modificat din momentul încărcării etc.).

Windows foloseşte paginarea la cerere, ceea ce înseamnă că paginile sunt încărcate în memoria primară când sunt cerute. În plus, PTE-urile nu sunt create până când pagina corespunzătoare nu este încărcată în memorie, în ideea că procesele nu trebuie să rezerve spații de memorie primară, pe care să nu le folosească. Din acest motiv, sistemul de operare trebuie să păstreze alte structuri de date care să reliefeze operațiile de rezervare şi de cerere de alocare.

Descriptorul de adrese virtuale(VAD – Virtual Address Descriptor) este creat atunci când un proces face o rezervare de adrese de memorie virtuală sau o cerere de astfel de spațiu. Când un fir de execuție referențiază pentru prima dată adrese din VAD, este creat PTE-ul şi astfel translatarea adreselor se desfăşoară normal.

Să ne reamintim...

Fiecărui proces sub Windows, îi este acordat un spațiu de adrese virtuale de dimensiune fixă (4 Go), care este mult mai mare decât dimensiunea memoriei primare a unui calculator.

Dimensiunile spațiilor de adrese virtuale utilizate de proces, respectiv de nucleul sistemului de operare diferă de la o variantă la alta de sistem Windows.

Translatarea adreselor se face cu ajutorul unor componente hardware, pentru a detecta rapid paginile de memorie virtuală ale procesului care lipsesc din memoria internă şi pentru a le pune în corespondență cât mai rapid pagini fizice. O adresă virtuală este memorată pe 32 de biți şi este generată de către procesor.

Înlocuiți zona punctată cu termenii corespunzători

1. Fiecare procesor are o anumită ....., adică dimensiunea minimă a blocului de adrese care va fi rezervat. De obicei, aceasta este de ..... De asemenea fiecare procesor are o anumită ..... a paginii, de obicei .....
2. Win2K foloseşte o schemă de adresare structurată pe ..... niveluri.
3. Cei mai puțin semnificativi ..... din cadrul ....., reprezintă ..... Restul de biți se numeşte ..... şi este separat în două părți, denumite ..... şi .....
4. Descriptorul de proces conține ..... la începutul ....., care are ..... de intrări, pentru procesul dat. ....., este un deplasament unde este localizată o intrare în .....
5. Intrarea în ....., care la rândul lor au câte ..... de intrări este găsită folosind indicele ..... din adresă, ca un index în tabela de pagini.
6. Deplasamentul este adăugat adresei ....., pentru a obține ..... căutat din memoria primară.

M6.U2.6. Administrarea proceselor sub Windows

Această componentă furnizează serviciile necesare pentru creare, ştergere şi utilizare a thread-urilor şi proceselor. Relațiile dintre procesele părinte şi cele fiu sunt rezolvate de către subsistemele de mediu.

Exemplu. Când o aplicație care este executată sub Win32 cere crearea unui proces, se execută următorii paşi:
- Se apelează o metodă `CreateProcess`, prin care este transmis un mesaj subsistemului Win32, care cere administratorului de procese să creeze acel proces.
- Administratorul de procese cere administratorului de obiecte să creeze un obiect proces şi să returneze un manipulator de obiecte lui Win32.
- Win32 va apela din nou administratorul de procese, pentru a crea threadul respectiv
- În final, Win32 returnează manipulatorul către noul thread.

Caracteristicile cele mai importante ale proceselor sub Windows sunt:
- sunt implementate ca obiecte.
- un proces în execuție poate sa conțină unul sau mai multe thread-uri.
- ambele au posibilități de sincronizare.

Figura 6.2.2., ilustrează modul în care un proces controlează sau utilizează resursele.

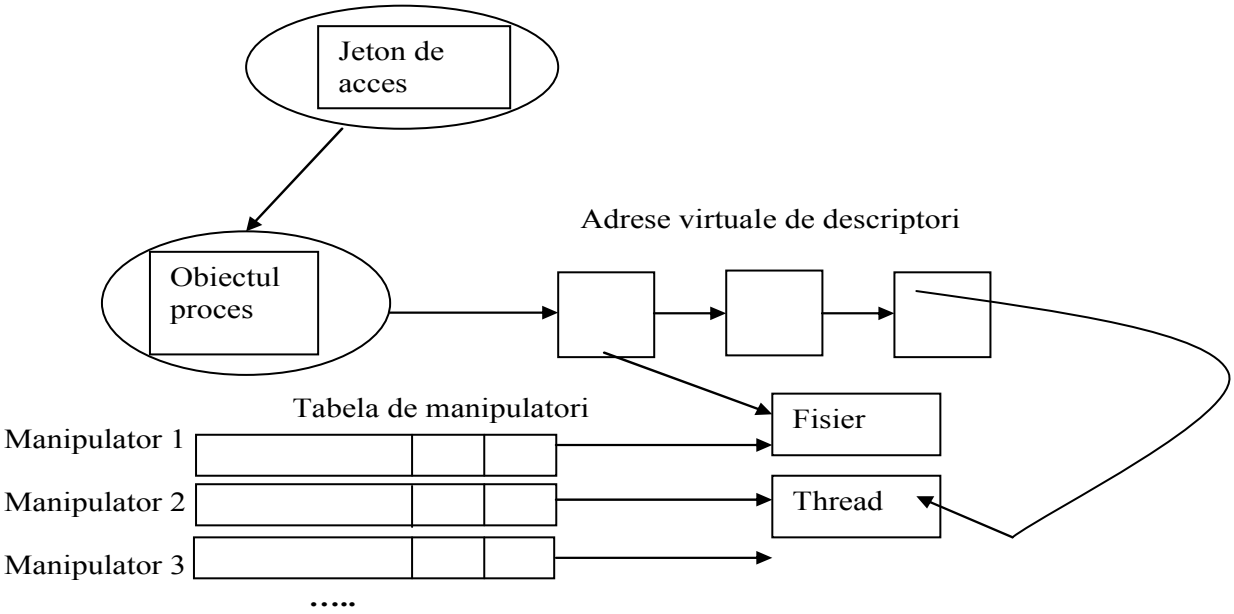


Figura 6.2.2. Modul în care un proces controlează sau utilizează resursele.

Fiecărui proces îi este asignat un jeton de acces, jetonul primar al procesului. Când un utilizator se loghinează pentru prima dată, Windows crează un jeton de acces care include identificatorul de securitate al utilizatorului. Fiecare proces care este creat sau executat în contextul acestui utilizator, are o copie a acestui jeton de aces. Windows foloseşte jetonul pentru a valida drepturile unui anumit proces, din clasa legată de utilizatorul respectiv, de a accesa obiecte securizate sau de a executa anumite funcții cu caracter restrictiv pe obiectele

securizate. De asemenea, jetonul de acces controlează dacă procesul îşi poate schimba propriile atribute; în acest caz, procesul nu are deschis un handler pentru a-şi accesa jetonul. Dacă procesul încearcă să deschidă un astfel de handler, sistemul de securitate determină dacă acest lucru este permis, adică procesul îşi poate schimba atributele proprii.

De asemenea, legat de procese există o serie de blocuri ce definesc spațiul de adrese virtuale asignat în mod curent unui anumit proces. Procesul nu poate modifica direct aceste structuri, dar poate cere acest lucru administratorului memoriei virtuale, care poate furniza un serviciu de alocare a memoriei pentru procese. Procesele includ un obiect tabelă (figura 6.22.), cu manipulatori pentru alte procese (thread-uri) vizibile acestui proces. În plus, procesele au acces la un obiect fişier şi la unul secțiune , care definesc o secțiune a memoriei partajate.

Obiectele proces si thread. Windows utilizează două tipuri de obiecte legate de procese: procese şi thread-uri. Procesul este o entitate care corespunde unui job al unui utilizator sau unei aplicații ce deține anumite resurse, cum ar fi memorie si fişiere deschise. Thread-ul este o componentă a procesului ce are un singur fir de control, care poate fi întrerupt, astfel încât procesorul poate comuta la alt thread. Fiecare proces Windows este reprezentat ca un obiect, a cărui structură generală este arătată în figura 6.2.2; de asemenea, el este definit printr-un număr de atribute şi încapsulează un număr de acțiuni sau servicii, pe care le poate executa. Un proces va executa un serviciu când apelează o mulțime de metode ale interfeței. Când Windows creazăun proces nou, el foloseşte un şablon pentru a genera o nouă instanță obiect.

La momentul crearii, sunt asignate valori ale atributelor. Atributele unui obiect proces sunt:

- **Identificatorul procesului.** O valoare unică prin care sistemul de operare identifică procesul.
- **Descriptorul de securitate.** Descrie cine a creat un obiect, cine şi cum poate accesa obiectul respectiv.
- **Baza de prioritate.** O valoare pe baza căreia se calculează prioritatea procesului sau a firului de execuție.
- **Limitele cotelor.** Valorile maxime alocate ale dimensiunilor memoriei virtuale, memoriei fizice şi timpului procesor.
- **Timp de execuție.** Timpul total în care toate thread-urile procesului au fost executate.
- **Contoare ale I/O.** Variabilele care înregistrează numărul şi tipul operațiilor de I / O, efectuate de firele procesului.
- **Contoare ale memoriei virtuale.** Variabile care înregistrează numărul şi tipurile de operații pe care firele procesului le-au efectuat cu memoria virtuală.
- **Porturi pentru excepții/depanare.** Canale de comunicare între procese, la care managerul proceselor trimite un mesaj, atunci când unul dintre firele procesului generează o excepție. În mod normal acestea sunt conectate la subsistemul de mediu, respectiv la cel de depanare.
- **Starea de ieşire**. Motivul terminării procesui.

Serviciile executate de un proces sunt: **Crearea** procesului, **Deschiderea** procesului, **Cererea** de informații despre proces; **Setarea** de date ale procesului; **Terminarea** procesului.
Un proces Windows trebuie sa conțină cel puțin un thread. Acel thread, poate crea alte thread-uri. Într-un sistem multiprocessor, mai multe thread-uri ale aceluiaşi proces pot fi executate în paralel. Anumite atribute ale threadului le înlocuiesc pe cele ale procesului. În acele cazuri, valoarea atributului thread-ului este dedusă (mostenită) din cea a procesului. Unul dintre atributele unui obiect thread este contextul. Această informație permite thread-urilor să poată fi suspendate şi apoi reluate. Prin modificarea contextului unui thread, atunci când el este suspendat, se poate modifica comportamentul acestuia.

Planificarea firelor de execuţie. Fiecare „thread” se poate afla într-una dintre stările: **ready**, **standby**, **run**, **wait**, **transition** şi **terminate**. Tranzitiile posibile efectuate de thread-uri sunt:
- ready→standby; firul de execuţie de prioritate cea mai înaltă trece în starea **standby**, adică el va fi următorul care va primi serviciile unui procesor. În sistemele multi-procesor, pot exista mai multe astfel de thread-uri, corespunzător numărului de procesoare ale sistemului. Dacă prioritatea procesului aflat în starea **standby** este suficient de înaltă, thread-ul aflat în execuţie pe acel procesor poate fi fortat de către cel aflat în **standby**. În caz contrar, thread-ul aflat în **standby** aşteaptă pâna când thread-ul aflat în execuţie se blochează sau îşi epuizează cuanta de timp.

- standby→run; când nucleul execută o schimbare de context, thread-ul aflat în **standby** intră în starea **run**. Un thread aflat în starea **run**, va sta în această stare până când se întâmplă una dintre următoarele situaţii:

- va fi forţat de către un alt fir de execuţie de o prioritate mai înaltă;
- cuanta lui de timp s-a epuizat;
- se termină execuţia lui;
- va cere execuţia unui apel de sistem (se blocheaza).

- run→ wait; un thread intră în starea **Wait** într-una dintre următoarele situaţii:
(1) este blocat pe un eveniment;

Exemplu. Efectuarea unei operaţii de I/O.

(2) aşteaptă în mod voluntar, în scopuri de sincronizare;
(3) un subsistem de mediu direcţionează suspendarea thread-ului
- wait→ ready ; Când condiţia pentru care a intrat în starea de aşteptare este satisfacută, thread-ul trece în starea **Ready**, dacă toate resursele de care are nevoie sunt disponibile.
- **wait→transition;** Un thread intră în starea **Transition** când, după o aşteptare nu are disponibile toate resursele cerute.

Exemplu. Stiva thread-ului nu mai are suficient spaţiu de memorie.

- transition→ready (acum sunt disponibile resursele necesare pentru a fi executat în continuare)
- run→ ready (forţarea CPU)
- run→terminate (terminarea execuţiei);thread-ul poate fi terminat de către el însuşi, de către un alt thread sau când se termină procesul părinte. El poate fi şters definitiv din sistem sau poate fi reţinut de către executiv pentru o reiniţializare viitoare.

Pentru a determina ordinea de execuţie, dispecerul sistemului foloseşte o schemă de prioritate pe 32 de niveluri. Priorităţile sunt împărţite în două clase:
- clasa aplicaţiilor în timp real - firele de execuţie care au o prioritate de clasă cea mai înaltă, cu numere cuprinse între 16 şi 31,
- „thread”-uri de clasă variabilă, au numere de clasă cuprinse între 0 şi 15.

Dispecerul foloseşte câte o coadă pentru fiecare nivel de prioritate şi atunci când se pune problema selectării unui nou fir de execuţie, acesta va traversa sistemul de cozi, de la cea de nivel cel mai înalt, către cea de nivel cel mai scăzut, până când găseşte un thread gata de execuţie (aflat în starea **standby**). Dacă nu este găsit nici un astfel de „thread”, dipecerul va executa un „thread” special, care îl va trece în stare de aşteptare(idle). De asemenea, când un thread gata de execuţie are o preferinţă pentru un anumit procesor, altul decât cel liber,

dispecerul va trece la un alt thread, aflat în starea **standby**. Când un „thread” în execuție, din clasa celor de prioritate variabilă își termină cuanta de timp alocată lui, prioritatea lui este scăzută, până la nivelul imediat, al „thread-ului” aflat în aşteptare cu nivelul cu prioritatea cea mai slabă.

Observații.
1. Această metodă, realizează o servire echitabilă a proceselor care utilizează mult CPU(orientate spre calcule). De asemenea, când un astfel de „thread” intră în starea **wait**, dispecerul recalculează prioritatea acestuia în funcție de tipul operației de I/O efectuate. Prin acest mod de lucru, în cazul thread-urilor interactive, se urmăreşte să se obțină un timp de răspuns rezonabil.
2. Planificarea poate să apară atunci când un thread intră într-una dintre stările **ready**, **wait** sau **finish** sau când trebuie schimbată prioritatea sau preferința către un anumit procesor ale acestuia.
3. Prin modul de organizare a planificării, se urmăreşte servirea prioritară a aplicațiilor în timp real. Deoarece acestea au priorități mai înalte, atunci când un astfel de thread este în starea **ready** şi toate procesoarele sunt ocupate, unul dintre threadurile de clasă variabilă va fi forțat să elibereze procesorul, care va servi aplicația în timp real.

Facilitatea de apel de procedură locală(LPC-**L**ocal **P**rocedure **C**all). Implementarea sistemului Windows foloseşte modelul client-server pentru implementarea subsistemelor de mediu. Modelul client-server este utilizat pentru a se implementa o multitudine de servicii pe care se spijină subsistemele de mediu, printre care: administrarea securității, utilizarea „spooling-ului” pentru tipărire, servicii de web, sistemul de fişiere partajate în rețea etc.

LPC este utilizată de sistemul de operare pentru:
- a transmite mesaje între un proces client şi un proces server, în interiorul aceluiaşi calculator.
- pentru comunicația între subsistemele Windows.

Observație. LPC este similar RPC (folosit pentru comunicația într-o rețea), dar LPC este optimizat pentru a fi folosit în interiorul unui sistem.

Procesul server face public un obiect port de conectare care este vizibil global. Când un proces client doreşte servicii de la un subsistem, el deschide un manipulator la acel port şi transmite o cerere de conexiune la acel port. Serverul creează un canal şi returnează un manipulator pentru acel canal: canalul constă dintr-o pereche de porturi de comunicație, unul pentru mesaje de la client la server iar celălalt de la server la client.

Când este creat un canal LPC, trebuie să fie specificată una dintre următoarele tehnici:
- Transmiterea mesajelor de dimensiune mică se realizează prin punerea mesajelor într-o coadă, de unde sunt luate de către procesele cărora le sunt destinate.
- Pentru transmiterea mesajelor de dimensiune mare, se utilizează o zonă de memorie partajată; în acest caz, este creat un obiect prin care sunt gestionați pointerii către mesaje şi dimensiunea lor. Aceste informații sunt utilizate de procesele care primesc mesajele respective să le preia direct.
- Utilizarea unei API prin care se poate citi/scrie direct din spațiul de adrese al unui proces.

Subsistemele de mediu reprezintă procese executate în mod utilizator, care permit execuția unor aplicații dezvoltate sub alte sisteme de operare(MS-DOS, Windows pe 16 biți şi 32 de biți). Fiecare subsistem furnizează o interfață utilizatorului, prin care se pot lansa aplicații realizate în cadrul sistemelor de operare respective.

Windows foloseşte subsistemul Win32 ca mediu principal de operare pentru lansarea proceselor. Când este executată o aplicație, Win32 apelează administratorul memoriei virtuale

pentru a încărca fişierul executabil, corespunzător aplicaţiei respective. Administratorul memoriei returnează un mesaj către Win32, prin care specifică tipul aplicaţiei, pe baza căruia este ales un anumit subsistem de mediu sub care se va executa procesul respectiv.

Subsistemele de mediu folosesc facilitatea LPC pentru a obţine serviciile nucleului pentru aplicaţiile lansate. Astfel, se asigură robusteţea sistemului, deoarece parametrii transmişi printr-un apel de sistem sunt verificaţi, înainte de a fi cerută o anumită rutină. De asemenea, o aplicaţie executată sub un anumit mediu, nu poate apela o rutină corespunzătoare altui mediu.

Să ne reamintim...

Administrarea proceselor sub Windows furnizează serviciile necesare pentru creare, ştergere şi utilizare a thread-urilor şi proceselor. Relaţiile dintre procesele părinte şi cele fiu sunt rezolvate de către subsistemele de mediu.

Fiecărui proces îi este asignat un jeton de acces, jetonul primar al procesului. Când un utilizator se loghinează pentru prima dată, Windows crează un jeton de acces care include identificatorul de securitate al utilizatorului.

Windows utilizează două tipuri de obiecte legate de procese: procese şi thread-uri. Procesul este o entitate care corespunde unui job al unui utilizator sau unei aplicaţii ce deţine anumite resurse, cum ar fi memorie si fişiere deschise. Thread-ul este o componentă a procesului ce are un singur fir de control, care poate fi întrerupt, astfel încât procesorul poate comuta la alt thread.

Fiecare „thread" se poate afla într-una dintre stările: ready, standby, run, wait, transition şi terminate.

LPC este utilizată de sistemul de operare pentru a transmite mesaje între un proces client şi un proces server şi pentru comunicaţia între subsistemele Windows.

Înlocuiţi zona punctată cu termenii corespunzători

1. Windows foloseşte jetonul pentru a valida ..... unui anumit ....., din clasa legată de ....., de a accesa ..... sau de a executa anumite funcţii cu ..... pe obiectele .....
2. Jetonul de acces ..... dacă procesul îşi poate schimba propriile .....
3. Firul de execuţie de prioritate cea mai înaltă trece în starea ....., adică el va fi .....care va primi serviciile unui .....
4. Când nucleul execută o ....., thread-ul aflat în ..... intră în starea .....
5. Un thread aflat în starea run, va sta în această stare până când se întâmplă una dintre următoarele situaţii:.....
6. Un thread intră în starea Wait într-una dintre următoarele situaţii.....
7. Pentru a determina ordinea de ....., ...sistemului foloseşte o schemă de ..... pe ..... Priorităţile sunt împărţite în două clase:.....
8. Când un „thread" în execuţie, din clasa celor de prioritate ..... îşi termină ..... alocată lui, prioritatea lui este ....., până la nivelul imediat, al „thread-ului" aflat în .....cu nivelul cu prioritatea .....
9. LPC este utilizată de sistemul de operare pentru:.....

M6.U2.7. Sistemul de fişiere sub Windows

Sistemul de fişiere sub MS-DOS se bazează pe tabela FAT. În cazul sistemului de fişiere cu FAT pe 16 biţi există o serie de dezavantaje: fragmentarea internă, limitarea dimensiunii la 2 Go etc. Odată cu apariţia tabelei FAT pe 32 de biţi, s-au rezolvat o parte din aceste probleme.

Odată cu Windows XP, apare sistemul de fişiere NTFS(**N**ew **T**echnology **F**ile **S**ystem); acesta oferă o serie de facilităţi:

- refacerea volumelor dupa caderea sistemului sau defectarea discurilor;
- securitatea datelor;
- fişiere de orice dimensiune;
- fluxuri de date multiple;
- compresia fişierelor etc.

Windows oferă suport şi pentru alte sisteme de fişiere, cum este de exemplu FAT.

Entitatea NTFS este **volumul;** un volum poate ocupa unul sau mai multe discuri logice. Sub NTFS, unitatea de alocare este **clusterul;** acesta este format dintr-un număr de sectoare (a cărui dimensiune este o putere a lui 2 şi de obicei este 512 octeţi). Dimensiunea clusterului este fixată la formatarea discului; implicit este:

- dimensiunea unui sector de disc, dacă dimensiunea volumului este mai mică de 512 Mo;
- 1 Ko pentru volume de până la 1 Go;
- 2 Ko pentru volume de până la 2 Go;
- 4 Ko pentru volume mai mari.

Clusterele alocate unui fişier nu trebuie să fie contigue; astfel, un fişier se poate fragmenta pe un disc. Un cluster poate avea cel mult 2^{16} octeţi. În mod curent, dimensiunea maximă permisă a unui fişier sub NTFS este de 2^{32} clustere (2^{48} octeţi). Deoarece dimensiunea clusterului sub NTFS este mult mai mică decât cea sub FAT, fragmentarea internă a discului este mult mai mică.

Pentru adresare pe disc, NTFS foloseşte numărul logic de cluster (LCN-**L**ogical **C**luster **N**umber). Utilizarea clusterelor NTFS, face alocarea spatiului pentru fişiere independentă de dimensiunea sectorului. Aceasta permite NTFS:

- să suporte uşor discuri nonstandard, care nu au dimensiunea sectorului de 512 octeţi;
- să suporte eficient discuri şi fişiere de dimensiune mare, folosind clustere mari

Observaţie. Eficienţa alocării este datorată numărului mai mic de clustere şi implicit a intrărilor în tabele.

Sistemul Windows defineşte fişierul ca un flux de octeţi, care nu poate fi accesat decât secvenţial. Pentru fiecare instanţă a unui fişier deschis, există un pointer memorat pe 64 de biţi către următorul octet care urmează să fie accesat. Când fişierul este deschis, pointerul este setat cu 0 şi după fiecare operaţie de scriere/citire a k octeţi, valoarea pointerului este mărită cu k. Ca şi celelalte resurse gestionate de sistemul Windows şi gestionarea fişierelor este orientată pe obiecte. Un fişier este un obiect care are mai multe atribute. Fiecare atribut este un flux de octeţi, care poate fi creat, şters, citit sau scris. Atributele standard (numele fişierului, data creerii, descriptorul de securitate etc.) sunt comune tuturor fişierelor, pe când anumite atribute particulare sunt specifice unei anumite clase de fişiere (director, de **exemplu**). Fiecare fişier sub NTFS este descris prin una sau mai multe înregistrări dintr-un tablou, memorat într-un fişier special, numit tabela MFT(**M**aster **F**ile **T**able), prezentată în figura 6.2.3.

Zona de bootare	Tabela MFT	Sistemul de fişiere	Zona de fişiere

Figura 6.2.3. Structura tabelei MFT

Organizarea informaţiilor pe un volum NTFS. Fiecare element de pe un volum este fişier şi fiecare fişier constă dintr-o colecţie de atribute (inclusiv datele conţinute în fişier sunt tratate ca un atribut). Din figura 6.2.3, observăm că un volum NTFS este format din 4 regiuni(zone).

Primele sectoare ale volumului sunt ocupate de catre **partiţia de boot** (pâna la 16 sectoare lungime); aceasta conţine:

- informaţii despre organizarea volumului şi structuri ale sistemului de fişiere;
- o secvenţăde cod care, încarcată în memorie şi lansată în execuţie va încarca sistemul de operare (fişierul de boot-are);
- informatii de "startup".

Tabela MFT conţine informaţii despre toate fişierele şi foldere-le (directoare) de pe acest volum NTFS. În esenţă, MFT este o listă a tuturor fişierelor şi atributelor lor de pe acest volum NTFS, organizată ca o mulţime de linii, într-o structură de bază de date relaţională. Dimensiunea înregistrării este cuprinsă între 1 Ko şi 4 Ko, fiind fixată la crearea sistemului. Atributele de dimensiune mică sunt memorate chiar în tabelă(atribute rezidente), iar celelalte sunt păstrate pe disc şi tabela MFT va conţine un pointer către ele. Fiecare înregistrare din MFT constă dintr-o mulţime de atribute care serveşte la definirea caracteristicilor fişierului (sau folder-ului) respectiv şi a conţinutului său. Aceste atribute sunt:

- **Informaţii standard**: atribute de acces (read-only, read/write, etc.); marca timpului (data creerii, data ultimei modificari a fisierului) ; contorul de legaturi la fişier.
- **Lista de atribute**: este utilizată pentru localizarea atributelor fişierului.
- **Numele fisierului**.
- **Descriptorul de securitate**: Specifică proprietarul fişierului şi cine are acces la el.
- **Date**: Conţinutul fişierului.
- **Indexul radăcinii**: folosită pentru a implementa foldere.
- **Informaţii despre volum**: versiunea şi numele volumului.
- **Harta de biţi**: Indică inregistrările utilizate.

După MFT este o regiune (de lungime 1 Mb), ce conţine **sistemul de fişiere**. Printre fişierele din această regiune sunt:

• **MFT2:** este folosit pentru a garanta accesul la MFT în cazul unei defecţiuni apărute pe un singur sector.
• **Fişierul de jurnalizare** (**Log file):** Conţine o listă a tranzacţiilor efectuate; este folosit pentru refacerea NTFS.
• **Harta de biţi a clusterelor:** Indică ce clustere sunt folosite.
• **Tabela de definire a atributelor:**defineşte tipurile de atribute permise pe acest volum; indica daca ele pot fi indexate şi dacă pot fi refăcute în timpul unei operaţii de refacere a sistemului.
• **Fişierul clusterelor defecte**: conţine adresele de pe disc a zonelor care nu pot fi utilizate.

Fiecare fişier dintr-un volum NTFS are un **identificator unic**, numit **referinţă** la fişier şi este reprezentată pe 64 de biţi, dintre care 48 sunt alocaţi **numărului de fişier**, iar ceilalţi 16 conţin **numărul de secvenţă**. Numărul de fişier este numărul înregistrării corespunzătoare fişierului din tabela MFT. Numărul de secvenţă este incrementat de fiecare dată când o intrare din MFT este reutilizată. Acest număr permite NTFS să execute anumite verificări, cum ar fi,

de **exemplu** referențierea unei intrări în tabelă, alocată unui alt fişier, după ce a fost utilizată pentru un fişier care a fost şters.

Ca şi celelalte sisteme de fişiere, spațiul de nume NTFS este organizat ierarhic. Structura arborescentă utilizată este arborele B+. Acesta are proprietatea că lungimea oricărui drum de la rădăcină la oricare dintre frunze are aceeaşi lungime. Arborele B+ este utilizat deoarece elimină costurile necesare reorganizării structurii de fişiere.

Rădăcina index a directorului, aflată la cel mai înalt nivel al arborelui, conține pointeri către celelalte elemente ale structurii. Fiecare intrare din director, conține numele şi o serie de informații despre fişier(timpul ultimei prelucrări, dimensiunea fişierului etc.), copiate din tabela MFT. Astfel, se poate lista conținutul oricărui director, fără a se consulta tabela MFT, care, datorită complexității acesteia, este o operație costisitoare.

Să ne reamintim...

Odată cu Windows XP, apare sistemul de fişiere NTFS care oferă o serie de facilități: refacerea volumelor dupa caderea sistemului sau defectarea discurilor; securitatea datelor; fişiere de orice dimensiune; fluxuri de date multiple; compresia fişierelor etc.

Entitatea NTFS este **volumul;** un volum poate ocupa unul sau mai multe discuri logice. Sub NTFS, unitatea de alocare este **clusterul;** acesta este format dintr-un număr de sectoare, a cărui dimensiune este o putere a lui 2. Dimensiunea clusterului este fixată la formatarea discului

Sistemul Windows defineşte fişierul ca un flux de octeți, care nu poate fi accesat decât secvențial. Pentru fiecare instanță a unui fişier deschis, există un pointer memorat pe 64 de biți către următorul octet care urmează să fie accesat. Când fişierul este deschis, pointerul este setat cu 0 şi după fiecare operație de scriere/citire a k octeți, valoarea pointerului este mărită cu k.

Fiecare fişier sub NTFS este descris prin una sau mai multe înregistrări dintr-un tablou, memorat într-un fişier special, numit tabela MFT(**M**aster **F**ile **T**able). Tabela MFT conține informații despre toate fişierele şi foldere-le (directoare) de pe acest volum NTFS.

Înlocuiți zona punctată cu termenii corespunzători

1. Dimensiunea clusterului este fixată la formatarea discului; implicit este:....., dacă dimensiunea volumului este mai mică de 512 Mo; ..... pentru volume de până la 1 Go; ..... pentru volume de până la 2 Go; ....pentru volume mai mari.
2. Un cluster poate avea cel mult ..... octeți. În mod curent, dimensiunea maximă permisă a unui fişier sub NTFS este de .....clustere (.....octeți).
3. Fiecare fişier sub ..... este descris prin una sau mai multe ..... dintr-un ....., memorat într-un fişier special, numit .....
4. MFT2 este folosit pentru a garanta ..... la MFT în cazul unei ..... apărute pe un singur .....
5. Fiecare fişier dintr-un volum NTFS are un ....., numit ..... la fişier şi este reprezentat pe ....., dintre care ..... sunt alocați ....., iar ceilalți ..... conțin .....

M6.U2.9. Test de evaluare a cunoştinţelor
Marcaţi varianta corectă.

1. Raţiunea pentru care a fost creată o singură API este:
   a) legată de necesitatea compatibilităţii cu aplicaţiile scrise sub sistemele de operare care respectă standardele POSIX. ☐
   b) legată de necesitatea portabilităţii între diferite sisteme Windows. ☐
   c) legată de necesitatea compatibilităţii cu „motoarele” de căutare pe Internet. ☐
   d) legată de posibilitatea conectării sistemelor de calcul, la reţelele de televiziune prin cablu. ☐
2. Care dintre niveluri realizează interfaţa între sistemul Windows şi componenta hardware a calculatorului:
   a) Nucleu ☐
   b) HAL. ☐
   c) Executiv. ☐
   d) Subsistemul de mediu. ☐
3. Planificarea firelor de execuţie este sarcina:
   a) Nucleului. ☐
   b) HAL. ☐
   c) Executivului. ☐
   d) Subsistemului de mediu. ☐
4. Codul componentelor sistemului Windows, care este dependent de un anumit procesor, se găseşte în:
   a) Fişiere DLL. ☐
   b) Nucleu. ☐
   c) Administratorul „Plug and Play”. ☐
   d) Administratorul I/O. ☐
5. Care dintre componentele Windows face restul sistemului de operare complet independent de hard:
   a) Nucleul. ☐
   b) Executivul ☐
   c) Administratorul I/O. ☐
   d) Administratorul securităţii sistemului. ☐
6. Firele de execuţie care au prioritatea cea mai înaltă sunt:
   a) Clasa aplicaţiilor în timp real. ☐
   b) Clasa aplicaţiilor aflate în starea Run. ☐
   c) Clasa aplicaţiilor aflate în starea standby. ☐
   d) Clasa aplicaţiilor aflate în starea Ready. ☐
7. Servirea echitabilă a threadurilor de clasă variabilă presupune:
   a) Scăderea priorităţii threadurilor din această clasă, atunci când îşi epuizează cuanta de timp, sub nivelul tuturor firelor de execuţie aflate în aşteptare. ☐
   b) Trecerea threadului respectiv în starea standby. ☐
   c)Creşterea priorităţii threadurilor din această clasă, atunci când îşi epuizează cuanta de timp, sub nivelul tuturor firelor de execuţie aflate în aşteptare. ☐
   d) Trecerea threadului respectiv în starea transition. ☐
8. Care dintre proceduri îndeplinesc funcţii de sistem şi sunt executate în mod supervizor:
   a) DPC. ☐
   b) LPC ☐
   c) APC. ☐
   d) MPC. ☐

9. Care dintre proceduri sunt executate în contextul unui anumit fir de execuție, pentru situații în care firul respectiv trebuie să opereze într-un anumit spațiu de adrese.

a) DPC. ☐ c) APC. ☐
b) LPC ☐ d) MPC. ☐

10. Care dintre proceduri sunt utilizate pentru a transmite mesaje, între un proces client şi un proces server:

a) DPC. ☐ c) APC. ☐
b) LPC ☐ d) MPC. ☐

11. Pentru obținerea serviciilor nucleului, subsistemele de mediu folosesc facilitatea:

a) DPC. ☐ c) APC. ☐
b) LPC ☐ d) MPC. ☐

12. Zonele de memorie nucleu utilizate de către un proces, au puncte de intrare diferite, față de zonele de memorie obişnuite, localizate în:

a) PDE ☐ c) KDE. ☐
b) PTE ☐ d) WTE. ☐

13. Dimensiunea clusterului pentru volume mai mari de 2 Go este de:

a) 1 Ko ☐ c) 4 Ko ☐
b) 2 Ko ☐ d) 8 Ko ☐

14.Dimensiunea maximă permisă a unui fişier sub NTFS este de:

a) 2^{16} octeți ☐ c) 2^{48} octeți ☐
b) 2^{32} octeți ☐ d) 2^{64} octeți ☐

M6.U2.10. Rezumat.

Versiunile Windows 1.0, 2.0, 3.0, 3.1 au fost lansate de Microsoft ca interfețe grafice(GUI), care se utilizau peste sistemul de operare MS-DOS. Windows NT este prima versiune de sistem de operare Windows, care este un sistem de operare pentru calculatoare pe 32 de biți, care suportă aplicațiile scrise DOS şi are aceeaşi interfață cu Windows 3.1. Windows 95 a reprezentat o variantă îmbunătățită a Windows 3.1. Windows NT 4.0 are aceeasi arhitectura interna ca şi versiunile Windows 3.x şi furnizează aceeaşi interfață utilizator ca şi Windows 95. Windows 98 a fost succesorul lui Windows 95. Versiunea 5.0 a NT a fost lansată în anul 2000 şi fost redenumită Windows 2000(Win2K). Windows XP este succesorul lui Windows 2000; el poate fi utilizat ca sistem de operare pentru calculatoarele legate în rețea, dar poate înlocui şi sistemele Windows 95/98/ME destinate calculatoarelor individuale. În 2007 apare Windows Vista, oal cărui biectiv principal a fost securitatea. Windows 7este varianta îmbunătățită a Windows Vista.

Windows înlocuieşte modul de lucru text cu cel grafic şi permite execuția simultană a mai multor lucrări.

Sistemele Windows sunt organizate pe niveluri. Principalele niveluri sunt cel de abstractizare a hardwarelui (HAL), nucleul şi executivul.

Administrarea proceselor sub Windows furnizează serviciile necesare pentru creare, ştergere şi utilizare a thread-urilor şi proceselor. Relațiile dintre procesele părinte şi cele fiu sunt rezolvate de către subsistemele de mediu. Fiecărui proces îi este asignat un jeton de acces, jetonul primar al procesului. Când un utilizator se loghinează pentru prima dată, Windows crează un jeton de acces care include identificatorul de

securitate al utilizatorului.

Windows utilizează două tipuri de obiecte legate de procese: procese şi thread-uri. Procesul este o entitate care corespunde unui job al unui utilizator sau unei aplicații ce deține anumite resurse, cum ar fi memorie si fişiere deschise. Thread-ul este o componentă a procesului ce are un singur fir de control, care poate fi întrerupt, astfel încât procesorul poate comuta la alt thread. Fiecare „thread" se poate afla într-una dintre stările: ready, standby, run, wait, transition şi terminate.

Odată cu Windows XP, apare sistemul de fişiere NTFS care oferă o serie de facilități: refacerea volumelor dupa caderea sistemului sau defectarea discurilor; securitatea datelor; fişiere de orice dimensiune; fluxuri de date multiple; compresia fişierelor etc. Entitatea NTFS este **volumul;** un volum poate ocupa unul sau mai multe discuri logice. Sub NTFS, unitatea de alocare este **clusterul;** acesta este format dintr-un număr de sectoare, a cărui dimensiune este o putere a lui 2. Dimensiunea clusterului este fixată la formatarea discului

Sistemul Windows defineşte fişierul ca un flux de octeți, care nu poate fi accesat decât secvențial. Pentru fiecare instanță a unui fişier deschis, există un pointer către următorul octet care urmează să fie accesat. Fiecare fişier sub NTFS este descris prin una sau mai multe înregistrări dintr-un tablou, memorat într-un fişier special, numit tabela MFT(**M**aster **F**ile **T**able). Tabela MFT conține informații despre toate fişierele şi foldere-le (directoare) de pe acest volum NTFS.

Răspunsuri/Indicații de rezolvare a testelor de evaluare

Modulul 1

1.a; 2.b; 3.c; 4b.; 5.c; 6.d; 7.c; 8.a; 9.b; 10.d.

Unitatea de învățare M2.U1

1.c; 2.c; 3.b; 4.a; 5.b; 6.b; 7.b; 8.a; 9.a; 10.c; 11.c; 12.a; 13.a.

Unitatea de învățare M2.U2

1.b; 2.a; 3.c; 4.d; 5.b; 6.b; 7.c; 8.a; 9.a.

Unitatea de învățare M2.U3

Problema 1.

i) **Indicație:** Se va verifica siguranța secvenței $(p_2, p_4, p_5, p_3, p_1)$.

ii) În cazul în care procesul p_2 formulează o cerere suplimentară pentru 1 element din r_1, respectiv 2 elemente din r_2, trebuie să se verifice dacă această cerere poate fi satisfăcută. Conform algoritmului bancherului, se verifică îndeplinirea relațiilor:

$C_2 \le N_2$, adică $(1,2,0) \le (1,2,2)$

$C_2 \le D$, adică $(1,2,0) \le (2,3,0)$

Cum inegalitățile sunt îndeplinite, se simulează alocarea. Starea sistemului devine:

Să considerăm un sistem cu 5 procese p_1, p_2, p_3, p_4, p_5 şi trei resurse r_1, r_2, r_3, care au $10, 5$ respectiv 7 elemente. Se consideră că starea inițială este definită de: $A = \begin{pmatrix} 0 & 1 & 0 \\ 3 & 2 & 0 \\ 3 & 0 & 2 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix}$,

$N = \begin{pmatrix} 7 & 4 & 3 \\ 0 & 0 & 2 \\ 6 & 0 & 0 \\ 0 & 1 & 1 \\ 4 & 3 & 1 \end{pmatrix}$, $D = \begin{pmatrix} 1 & 1 & 0 \end{pmatrix}$.

Indicație: Se va verifica siguranța secvenței $(p_2, p_4, p_5, p_1, p_3)$.

iii) **Indicație:** Se consideră permutările mulțimii $\{1,2,3,4,5\}$ şi se verifică siguranța secvenței corespunzătoare cu algoritmul bancherului.

iv) Nu sunt îndeplinite inegalitățile $C_5 \le N_5$ şi $C_5 \le D$.

Problema 2.

i) Se consideră secvența (p_1 p_3 p_2 p_4 p_5)

Pas 1. $T=(0,0,0,0,0)$; $L=(2,3,0)$.

Paşii 2 şi 3.

Iterația 1. Pentru procesul p_1 avem $\begin{pmatrix} 0 & 0 & 0 \end{pmatrix} \le \begin{pmatrix} 2 & 3 & 0 \end{pmatrix}$, deci $L := \begin{pmatrix} 2 & 4 & 0 \end{pmatrix}$, $T=(1\ 0\ 0\ 0\ 0)$

Iterația 2. Pentru procesul p_3 avem $(0\ 0\ 0) \le (2\ 4\ 0)$, deci $T=(1\ 0\ 1\ 0\ 0)$ $L=(5\ 3\ 3)$.

Iterația 3. Pentru procesul p_2 avem $(2\ 0\ 2) \le (5\ 3\ 3)$, deci $T=(1\ 1\ 1\ 0\ 0)$ $L=(7\ 3\ 3)$.

Iterația 4. Pentru procesul p_4 avem $(2\ 1\ 1) \le (7\ 3\ 3)$, deci $T=(1\ 1\ 1\ 1\ 0)$ $L=(8\ 4\ 4)$.

Iterația 5. Pentru procesul p_5 avem $(0\ 0\ 2) \leq (8\ 4\ 4)$, deci $T=(1\ 1\ 1\ 1\ 1)$ $L=(8\ 4\ 6)$.

Pas 4. $T=(1\ 1\ 1\ 1\ 1)$, deci nu există interblocare în sistem.

i) $C = \begin{pmatrix} 0 & 0 & 0 \\ 2 & 0 & 2 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 2 \end{pmatrix}$, se consideră secvența $(p_1\ p_4\ p_2\ p_3\ p_5)$.

Problema 3.

i) $P=\{p_1,p_2,p_3\}$; $R=\{r_1,r_2,r_3,r_4\}$;
$A=\{(p_1,r_1),(r_1,p_2),(r_2,p_2),(r_2,p_1),(p_2,r_3),(r_3,p_3)\}$

ii) r_1 are un element, alocat procesului p_2; r_2 are două elemente, unul alocat procesului p_1 şi unul alocat procesului p_2; r_3 are un element, alocat procesului p_3.

iii) p_1 are alocat un element din r_2 şi cere r_1; p_2 are alocate câte un element din r_1 şi r_2 şi cere r_3; p_3 are alocat un element din r_3.

iv) p_3 cere r_2.

Problema 4.

În graful din figura 2.3.6 se adaugă (p_3,r_4) şi (r_4,p_2).

Problema 5.

i) Mulțimea arcelor este: $\{(p_1,p_2),(p_2,p_3),(p_2,p_4),(p_2,p_5),(p_3,p_4),\ (p_4,p_1)\}$

ii) Avem circuitul $((p_1,p_2),(p_2,p_3),(p_3,p_4),(p_4,p_1))$, deci există interblocare.

Unitatea de învățare M3.U1

I. 1.b; 2.a; 3.d; 4.b; 5.c; 6.a; 7.b; 8.c.

II.

first-fit

proces 1 (138 Ko) - partiția 1 (150 Ko)
proces 2 (477 Ko) - partiția 2 (520 Ko)
proces 3 (192 Ko) - partiția 3 (230 Ko)
proces 4 (396 Ko) - partiția 5 (620 Ko)

best-fit

proces 1 (138 Ko) - partiția 1 (150 Ko)
proces 2 (477 Ko) - partiția 2 (520 Ko)
proces 3 (192 Ko) - partiția 3 (230 Ko)
proces 4 (396 Ko) - partiția 5 (620 Ko)

worst-fit

proces 1 (138 Ko) - partiția 5 (620 Ko)
proces 2 (477 Ko) - partiția 2 (520 Ko)
proces 3 (192 Ko) – partiția obținută după alocarea procesului 1
proces 4 (396 Ko) – trebuie să aştepte eliberarea unei partiții.

Se calculează spațiul neutilizat, datorită fragmentării, adică suma diferențelor dintre dimensiunea partiției şi spațiul utilizat de procese.

Unitatea de învățare M3.U2

1.c; 2.d; 3.b; 4.c; 5.b; 6.c; 7.b; 8.a; 9.a; 10.b; 11.a; 12.c.

Unitatea de învățare M4.U1

1.c; 2.b; 3.a; 4.c; 5.a; 6.a; 7.a; 8.d; 9.b; 10.b; 11.c; 12.a.

Unitatea de învățare M5U1
1.b; 2.c; 3.d; 4.b.; 5.a; 6.d; 7.a; 8.a; 9.a; 10.c.

Unitatea de învățare M6U1
1.c; 2.d; 3.d; 4.d; 5.b; 6.a; 7.c; 8.b; 9.b; 10.d; 11.c; 12.b ; 13.d; 14.a.

Unitatea de învățare M6U2
1.b; 2.b; 3.c; 4.a; 5.a; 6.a; 7.a; 8.a; 9.c; 10.b; 11.b; 12.b; 13.c; 14.c.

Bibliografie

1. Boian F.M. *Sisteme de operare interactive*, Editura Libris, Cluj-Napoca,1994
2. Boian F.M. . s.a *Sisteme de operare* Editura RISOPRINT, Cluj-Napoca, 2006
3.Florea I. *Sisteme de operare- Concepte fundamentale* Editura Universitatii "Transilvania", Brasov, 2004
4. Ionescu T. s.a *Sisteme de operare. Principii si functionare*, Editura Tehnica, Bucuresti, 1997
5. Jurca I. D. *Sisteme de operare*, Editura de Vest, Timisoara, 2001
6. Nutt G. *Operating Systems. A Moderne Perspective*. *Second Edition,*Addison Wesley Longmann, 2001
8. Silberschatz A. ş.a *Operating Systems Concepts.Seventh Edition*, Wesley Publishing Company, 2006
9. Stallings W. *Operating Systems: Internals and Design Principles, 6/E***,** Prentice Hall, 2009
10. Tanenbaum A. *Modern Operating Systems. Third Edition*, Pretince Hall, 2008