Editarea Legaturilor si Incarcarea Programelor

1. Activitatile realizate de editorul de legaturi

Operatia de editare contribuie la conceperea de programe structurate si complexe.

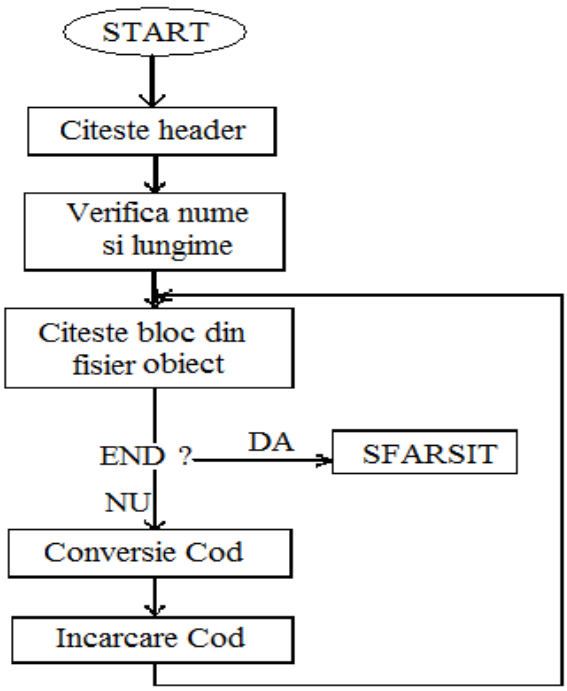
Un link editor primeste ca sursa unul sau mai multe fisiere obiect generate cu ajutorul compilatorului, sarcina sa fiind combinarea informatiilor din acestea pentru a obtine o imagine executabila valida (programul propriu-zis). Trebuie precizat faptul ca unele medii de dezvoltare inglobeaza functionalitatea link editorului in compilator.

Incarcarea reprezinta trecerea in memoria operationala a uni program asamblat si gata de executie. In unele cazuri incarcare presupune doar copierea unor date de pe disc in memoria operationala iar in altele implica alocarea unor zone de memorie, setarea unor biti de protectie.

1.1 Incarcarea de cod absolut (nerelocatabil)

Translateaza codul obiect din fisierul obiect in codul obiect din memorie. Codul obiect trebuie sa aiba un fomat cu inceput, cuprins, si sfarsit.

Incarcatorul preia bucati de cod si le salveaza in memorie. Pe suportul magnetic se va afla imaginea in binar. Incarcarea se face intr-un singur pas, conform figurii :



1.2 Incarcarea programelor relocatabile

Vom avea doi pasi. Sunt necesare urmatoarele structuri:

ESTAB – in acest tabel se gasesc variabilele externe cu numele si adresa si de asemenea toate referintele PUBLIC.

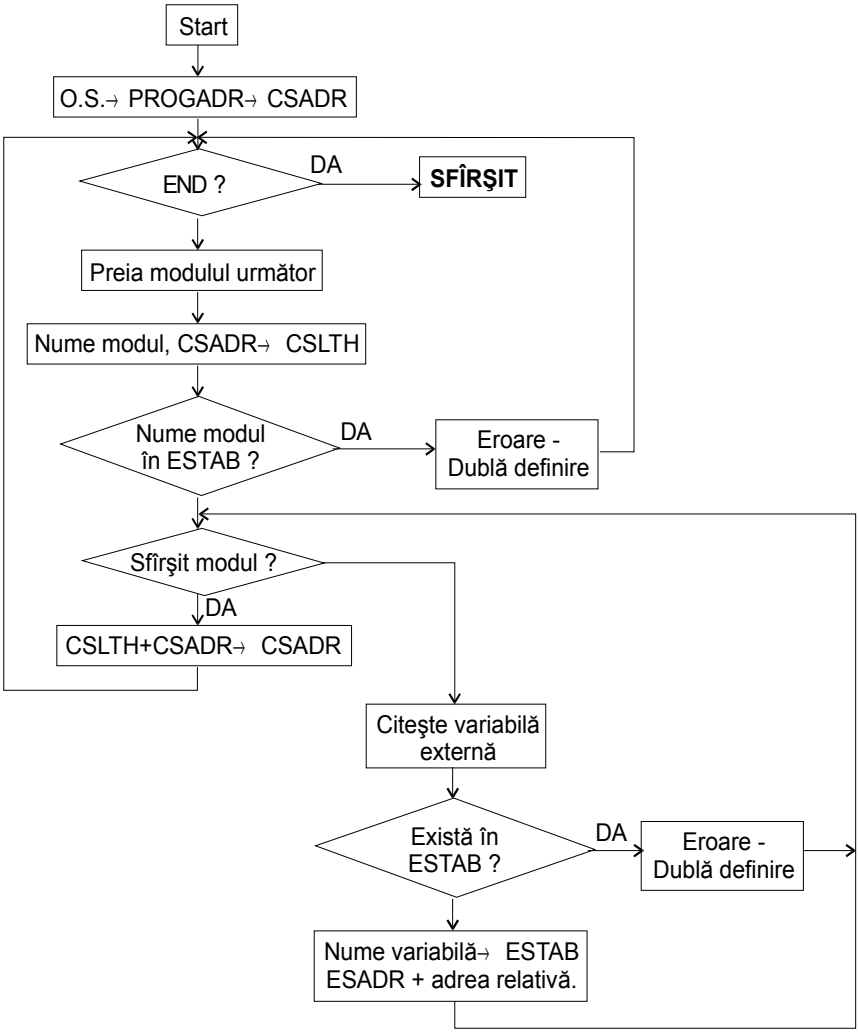
PROGADDR - adresele de incarcare a programelor legate;

CSADDR - adresa de inceput pentru modulul curent ce se prelucreaza si care contine referinta pentru relocatarea adreselor relative din interiorul modulului.

CSLTH - contine lungimea modulului curent.

EXECADDR - adresa de unde se executa programul.

PROGADDR - contine adresa de startare.



Relocatarea

Relocatarea este proprietatea programelor obiect de a fi executate in urma incarcarii lor la o alta adresa decat cea initiala. In modulele relocatabile adresele se calculeaza ca deplasamemte fata de o adresa de referinta(cea a primei instructiuni). In interiorul programele relocatabile intalnim doua tipuri de adrese : adrese absolute si adrese modificabile. In momentul incarcarii in memorie a unui modul relocatabil se opereaza asupra tuturor adreselor modificabile : se aduna valoarea adresei primei locatii date modulului de catre sistemul de operare.

Avantajele relocatarii sunt:

-posibilitatea creeri de programe complexe din mai multe module specializate.

-posibiliatea unui program de a rula din zone diferite de memorie libere la un moment dat.

Editarea de legaturi este extrem de utila pentru constructia de programe complexe dar in acelasi timp compacte prin alaturarea mai multor programe simple. Un astfel de program alcatuit din module relocatabile devine si el relocatabil, putand fi incarcat de sistemul de operare la o adresa de memorie convenabila.

Editarea de legaturi se face inaintea incarcarii si se genereaza astfel un modul relocatabil prin reuniunea mai multor module individuale si cu respectarea legaturilor Public si Extern.

Activitatile realizate de editorul de legaturi si de incarcator

1.3 Cautarea automata in biblioteci de module obiect.

Biblioteca este o multime organizata de module obiect elaborate professional care rezolva probleme dintr-un anumit domeniu de aplicatii. Exista doua tipuri de biblioteci:

- subprograme standard ce pot simplifica munca programatorilor.

- subprograme standard ce se adreseaza unei anumit domeniu.

Folosirea acestor biblioteci ofera o foarte mare flexibilitate la schimbarile de structura si pastreaza compatibilitatea programelor de aplicatie proprii, de utilizare generala.

Accesul la modulele obiect din cadrul bibliotecilor se face prin adaugarea la inceputul programului, ca variabila externa, a numelui modulului dorit din bibliotecile disponibile. Este recomandata specificarea la inceputul programului a numelui bibliotecilor dorite, pentru o mai mare claritate.

Mecanismul de acces la bibliotecile de module obiect devine activ dupa ce sa facut asamblarea si legarea programului, cand pot ramane in ESTAB variabile nesatisfacute. In acest caz editorul de legaturi cauta in bibliotecile specificate variabilele nedefinite. Avand in vedere ca apelul unei variabile dintr-o biblioteca poate declansa un apel in cascada exista posibilitatea ca una din biblioteci sa lipseasca. Atunci variabila va fi semnalata cu eroare de definire in ESTAB.

Timpul de cautare in biblioteci poate fi micsorat prin sortarea prealabila a modulelor obiect componente. Sortarea are la baza alcatuirea de cataloage de chei ce premit gasirea foarte rapida a numelui cautat. Listele de chei sunt rezidente in memoria operativa pe durata operatiunii de legare a modulelor obiect.

1.4 Conducerea procesului de incarcare

Structurarea programelor sursa se poate realiza pe trei niveluri ierarhice :

- sectiunea

- segmentul

- programul

Secctiunea:

Sectiunea - unitate de program independenta alcatuita dintr-o secventa de date, ce include atat definitii cat si instructiuni. Acestea pot fi folosite si in afara sectiunii unde au fost definite si atunci se vor numi simboluri externe si vor fi memorate in dictionarul de legaturi al sectiunii. Mai exact se memoreaza numele si adresele relative fata de inceputul sectiunii in care au fost definite

Dictionarul de legaturi se generat atunci cand sectiunea este compilata, permitand schimbul de informatii intre diferite sectiunii ale unui program. Informatiile pot reprezenta apeluri la simbolurile externe, sau definitii de simboluri externe.

Unul dintre avantajele oferite de folosirea sectiunilor este acela ca mai multi programatori pot scrie sectiunile uni program, ce pot fi mai apoi compilate separat pentru a obtine modulele obiect ale programului.

Segmentul

Segmentul este format din mai multe sectiuni ce au toate legaturile rezolvate. Este un modul obiect relocatabil format din sectiuni puse impreuna. Un segment este definit prin nume si prin adresa de intrare (adresa primei instructiuni executabile).

Pentru a realiza un segment, editorul de legaturi :

- defineste complet dictionarul de legaturi a fiecarei sectiuni : pentru fiecare simbol extern verifica daca exista o intrare intr-un alt dictionar de legaturi si memoreaza adresa acestuia

- aloca segmentului o zona continua de memorie, prin alocarea de zone de memorie continue si succesive tuturor sectiunilor ce alcatuiesc segmentul; astfel se determina adresele relative (la adresa 0 a segmentului) de incarcare a sectiunilor in memorie;

- relocateaza adresele simbolurilor externe, prin adaugarea la aceste adrese a adresei de incepu a sectiunii unde sunt definite.

Programul

Este o structura arborescenta formata din segmente, dintre care unul este segmentu radacina (principal) , celelalte fiind subordonate acestuia. Segmentele pot fi subordonate si unul fata de altul, rapotrurile de subordonare fiind determinate de ordinea de executie a acestora. Segmentele aceluias nivel ierarhic nu comunica intre ele si pot fi executate in paralel.

Segmentarea programului reprezinta procesul de impartire a programului in mai multe segmente, pentru a face posibila incarcarea segmentului radacina in memoria fizica in timpul executiei programului, restul segmentelor putand fi incarcate ulterior. Numarul segmentelor incarcate simultan in memorie depinde de memoria disponibila cat si de relatiile de subordonare dintre segmente.

In timpul generarii programului executabil, editorul de legaturi construieste si tabelul de legaturi asociat ce contine:

- numele segmentului radacina si a celorlalte segmente

- adresele relative (la adresa 0H a programului) ale segmentelor

- lungimea segmentului radacina si a celorlalte segmente

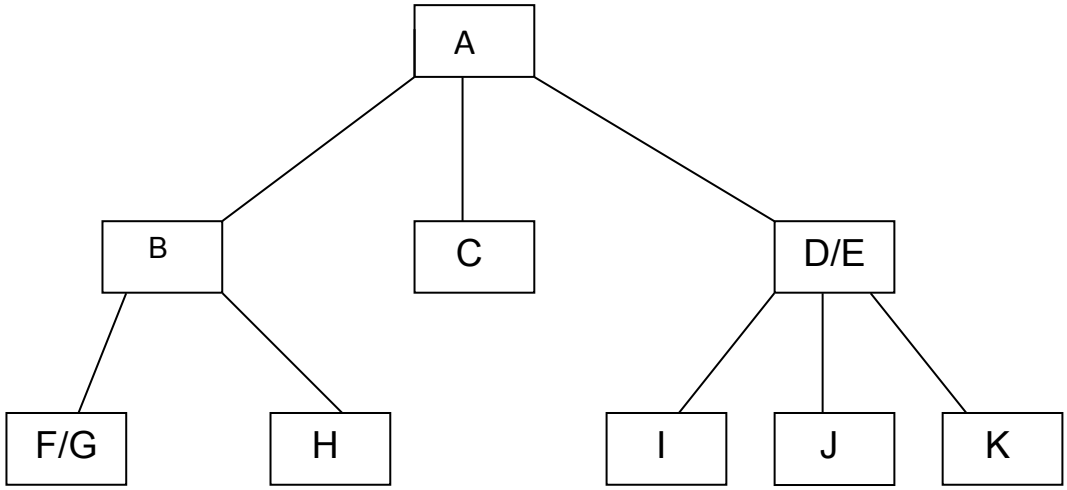
- adresa primei instructiuni a programului (adresa de intrare in program)

1.5 Structurarea in overlay a programelor.

Motivul pentru care este necesara structurarea in overlay a programelor este disponibilitate de spatiu de memorie. Intotdeauna vor fi aplicatii care depasesc capaciatea de memorie.

Aceasta problema se rezolva prin suparpunerea segmentelor in spatial disponibil si executarea lor in momente de timp diferite.Pentru gestiunea structurilor suprapuse (conducerea procesului de incarcare,executia,reluarea controlului pentru alta executie,etc) se foloseste un program general de coodonare .

Figura urmatoare prezinta un program executat prin suprapunerea de segmente :



Structuri ierahice de programe overlay

In acest arbore ierahic modulul A este radacina si acesta poate apela modulele B,C si D/E (acest modul desemneaza doua programme strans legate sau care functioneaza simultan).

Structurile ierarhic superioare trebuie sa fie incarcate inainte pentru gestiunea segmentelor inferioare. Segmentul radacina este incarcat la inceputul executiei si ramane acolo pana la stasitul acesteia. Segmetul parinte incarca segmentele fii si stabilind si ordinea incarcarii acestora deoarece pot fi mai multe reveniri.Segmentele de pe acelasi nivel ierarhic nu pot sa fie simultam in memorie ,iar cand se elibereaza un segment locul sau este luat de un segmnet de pe acelasi nivel.

Operatiile referitoare la prelucrarea structurilor suprapuse sunt indeplinite de un agent, manager al programelor supraincarcate, *overlay manager*, OVLMNG..Memoria operativa este ocupata de configuratii successive de segmente care coabiteaza in diferite faze de desfasurare a executiei.

Activitatea OVLMNG include:

- o zona rezervata gestiunii interne a programului OVLMNG;
- o tabela a segmentelor pentru gestiunea SEGFILE, care cuprinde la fiecare intrare:
    - nume de segment ;
    - indicator de actiune (exista / nu exista in memorie);
    - pozitie in structura (precizarea locului in ierarhie);
    - nivelul ierarhic al segmentului;
    - adresa de inceput ;
    - adresa de iesire;
    - locul segmentului in SEGFILE.

Functionare normala presupune ca segmentul la care se solicita transferul sa exite in memorie,in acest caz OVLMNG nu inetrvine in defasurare programului (apelurile se satisfac in interiorul aceluiasi modul).In cazul in care segmentul nu exista in memoria operativa OVLMNG stabileste ce zona din memoria operativa se elibereaza pentru a se face transferul segmentului dorit ,informatia despre identitatea segmentului obtinandu-se din zona proprie de date . OVLMNG actualizeaza adresele din structura de date proprie ,rezidenta in memorie si preda controlul la adresa de intrare in segmentul nou incarcat.Apelul solicitat se rezolva imediat segmentul fiind la locul sau in memoria operative.

1.6 Incarcarea sistemelor de operare :

Boot-are pentru Windows :

Aceasta faza variaza in functie de platforma. Din moment ce pasii initiali nu sunt specifici pentru sistemele de operare, procesul de boot-are se considera ca porneste in functie de processor astfel:

- pentru x86 sau x64: cand partitia sectorului de boot este executata in mod real si incarca NTLDR;

- pentru IA-64: cand pogramul IA64ldr.efi este executat ;

Din acest punct procesul de boot-are continua dupa cum urmeaza:

Un fisier NTLDR, localizat directorul root al discului de boot, este compus din 2 parti:

- modulul StartUp ;

- incarcator-ul sistemului de operare(osloader.exe) – aceste parti sunt stocate in acelasi fisier;

Cand fisierul NTLDR este incarcat in memorie si controlul este mai intai dat modulului de StartUp, procesorul functioneaza in modul real. Principala sarcina a modulelor de StartUp este de a comuta procesorul in modul protejat, care faciliteaza accesul la memorie pe 32 de biti, desi permite crearea initiala a Tabloului desriptorului de intreruperi, Tabloul pentru descriptori globali, a paginilor de tablou si activeaza paginarea . Aceasta ofera un mediu de operare de baza pe care sistemul de operare va fi construit. Apoi modulul StartUp incarca si lanseaza incarcator-ul sistemului de operare.

Incarcator-ul NTLDR al sistemului de operare include functionalitatea de baza de a accesa discuri prin interfata IDE formatate pentru sistemele de fisiere NTFS sau FAT. Discurile sunt accesate prin sistemul BIOS, prin rutine proprii ARC(Advanced RISC Computing) pe sistemele ARC sau prin retea folosind protocolul TFTP.

Ar trebui observat ca toate apelurile BIOS sunt facute in modul virtual 8086 dincolo de acest punct, pentru ca BIOS nu poate fi accesat direct in modul protejat. Daca discul de boot este un disc SCSI, un fisier aditional, Ntbootdd.sys, este incarcat pentru a se ocupa cu accesul la disc in locul rutinelor de baza.

Incarcatorul de boot citeste apoi continutul fisierului boot.ini pentru a localiza informatia in partitia sistemului. Daca fisierul boot.ini lipseste, incarcatorul de boot va incerca sa localizeze informatia din directorul standard de instalare. Pentru masinile Windows NT va incerca sa boot-eze din C:\WINNT. Pentru masinile Windows XP si 2003 va boota din C:\WINDOWS.

In acest moment ecranul este eliberat si, in versiunile Windows 2000 sau in versiunile ulterioare pentru NTLDR si IA64ldr care suporta hibernarea sistemului, partitia initiala a directorului root asa cum este definita in fisierul boot.ini se cauta un fisier de hibernare, hiberfil.sys. Daca acest fisier este gasit si un set activ de memorie este gasit in el, continutul acestui fisier va fi incarcat in memorie si controlul este transferat kernel-ului Windows-ului in acest punct din care hibernarea poate fi reluata. Fisierul este apoi marcat ca inactiv, astfel incat o “prabusire” a sistemului nu poate cauza ca acesta stare a memoriei sa fie reincarcata. Daca

incarcarea unei stari nu reuseste, atunci data viitoare cand NTLDR ruleaza va intreba utilizatorul daca vrea sa incerce o noua incarcare sau daca vrea sa “arunce” fisierul si sa continue cu procesul normal de boot-are.

Daca fisierul boot.ini contine mai mult de o intrare in sistemul de operare, atunci un meniu de boot-are este afisat pentru utilizator, permitandu-i sa aleaga ce sistem de operare sa fie incarcat. Daca un sistem de operare care nu este bazat pe NT, cum ar fi Windows98, este selectat, atunci NTLDR-ul incarca fisierul sectorului de boot asociat care este afisat in boot.ini si transfera executia catre acesta.

Daca este selectat un sistem de operare bazat pe NT, atunci NTLDR ruleaza ntdetect.com care aduna informatii de baza despre hardware-ul computerelor asa cum este declarat de BIOS.

In acest punct in procesul de boot-are , NTLDR elibereaza ecranul si afiseaza o bara de tip text care indica progresul . Windows 2000 de asemenea afiseaza textul “Starting Windows”. Daca utilizatorul apasa pe F8 in aceasta faza, atunci meniul avansat de boot-are este afisat, continand diferite moduri de boot-are (ex. Safe Mode).

Odata ce un anumit mod de boot-are a fost selectat(sau daca nu a fost apasat F8) procesul de boot-are continua. Daca versiunea de Windows pe 64 de biti este boot-ata (Windows XP Professional x64 Edition or Windows Server 2003 x64 Editions) atunci procesorul foloseste adresarea pe 64 de biti.

In continuare ,kernelul Windows *Ntoskrnl.exe* *si* Nivelul de Abstractizare a Hardware-ului (Hardware Abstraction Layer) *hal.dll* sunt citite in memorie. Daca unul din aceste fisiere nu se incarca, urmatorul mesaj este afisat catre utilizator "Windows could not start because the following file was missing or corrupt" si procesul de boot-are este intrerupt.

Daca mai multe configuratii sunt definite in registre, utilizatorul este intrebat la boot-are sa aleaga unul dintre ele.

Avand kernel-ul in memorie ,se incarca driverele dispozitivelor necesare la momentul boot-arii (dar nu se initializeaza) . Aceasta informatie este inregistrata in sectiunea `HKLM\SYSTEM a regitstrelor sistemului , intr-un set de chei de registre numit Control Set. HKLM\SYSTEM` contine: `ControlSet001`, `ControlSet002`, etc., ca si `CurrentControlSet`.

In timpul operarii obisnuite,Windows-ul foloseste `CurrentControlSet` pentru a scrie si citi informatii. `CurrentControlSet este o referinta catre unul dintre control set-uri memorate in registre. Windows-ul alege control set-ul folosit bazandu-se pe valorile setate in cheia HKLM\SYSTEM\Select`:

- Default este optiunea initiala a NTLDR-uli sau a IA64ldr daca nimic nu o suprascrie .
- Daca valoarea cheii Failed este aceeasi cu a cheii Default atunci NTLDR or IA64ldr afiseaza un mesaj de eroare ,indicand ca boot-area precedenta nu a reusit, si ii da utilizatorului optiunea de a incerca sa boot-eze fortat sau sa foloseasca ultima configuratie utilizabila "Last Known Good Configuration"
- Daca utilizatorul alege "Last Known Good Configuration" se va folosi control set-ul indicat de cheia LastKnownGood si nu Default.

Cand un control set este ales, cheia `Current` se modifica corespunzator. Cheia `Failed` este aceeasi cu cheia `Current` pana la sfatsitul procesului de boot-are. Atunci cand boota-rea se incheie cu success cheia `LastKnownGood` devine `Current`.

Driverele “Boot” sunt ,aproape exclusiv, drivere pentru controlerele de unitati de disc si pentru sistemele de fisiere (ATA, SCSI, etc), cu alte cuvinte sunt driverele absolut necesare ca ntoskrnl.exe sa porneasca incarcarea altor drivere si restul sistemului de operare. Driverele “Sistem ” acopera o gama larga de functiuni ale nucleului, incluzand driverul pentru afisare, pentru support CD-ROM,si stiva TCP/IP.

Sistemul de fisiere necesar pentru tipul partitiei (NTFS, FAT, or FAT32) unde se gaseste instalarea Windows-ului este de asemena incarcat.

Cu acestea incheiate, controlul este pasat de la NTLDR or IA64ldr catre kernel. In acest moment Windows NT afiseaza faimosul "ecran albastru" care afiseaza numarul de CPU-uri, cantitatea de memorie instalata, in timp ce Windows 2000, XP si 2003 comuta intr-un mod grafic afisand logo-ul Windows.

Faza de incarcare a Kernelului:

Initializarea kernelului si a subsistemelor Windows Executive se face in doi pasi.

In prima faza sunt create structuri de memorie interna „de baza" si se initializeaza controlerul de intreruperi pentru fiecare UCP (Unitate Centrala de Procesare). Managerul de memorie este initializat, creând zone de memorie tampon pentru sistemele de fisiere. Managerul de Obiecte lanaseaza jetonul de securitate initial adresat primului proces din sistem si Managerul de Procese (Process Manager) este lansat. In aceasta moment este creat procesul de sistem „in asteptare" (system idle process) cat si procesul sistemului insusi.

In cea de a doua faza sunt initializate driverele dispozitivelor ce au fost identificate de catre NTLDR ca fiind drivere de sistem.

De-a lungul procesului de incarcare a driverelor dispozitivelor, o bara de „evolutie" este vizibila in partea de jos a ecranului, pentru Windows 2000; pentru Windows XP aceasta a fost inlocuita de o bara animata care nu evidentiaza progresul real. Inainte de Windows XP aceasta faza a procesului de boot-are dura mai mult deoarece driverele erau initializate unul cate unul; spre deosebire de Windows XP unde driverele sunt toate initializate asincron.

Managerul de Sesiune (Session Manager)

Dupa ce toate driverele de sistem si de boot au fost incarcate, kernelul porneste subsistemul denumit Managerul de Sesiune (smss.exe).

Inainte ca vreun fisier sa fie deschis, Autochk este pornit de catre smss.exe. Autochk incarca toate partitiile si le verifica, una cate una, daca au fost inchise corespunzator inainte. In caz contrar, va lansa automat chkdsk; totusi utilizatorul poate opri acest proces prin apasarea oricarei taste inainte cu 10 secunde(aceasta caracteristica a fost introdusa de Windows NT 4.0 Service Pack 4, in versiunile anterioare nu puteai trece peste chkdsk).

La momentul boot-arii Managerul de Sesiune realizeaza urmatoarele functii:

- creaza variabile de mediu(HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Environment)

- porneste partea kernel a subsistemul Win32(win32k.sys). Acest lucru permite Windows-ului sa treaca in modul grafic

- porneste partea utilizator a subsistemului Win32 (csrss.exe).

- creaza fisierele responsabile cu paginarea memoriei virtuale(HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management)

- se realizeaza toate operatiunile de redenumire aflate in asteptare. Acest lucru permite inlocuirea fisierelor ce s-au aflat in folosinta pnetru realizarea unui reboot.

- porneste Windows Logon Manager (winlogon.exe). Winlogon este responsabil cu managementul log-arilor interative la un sistem Windows (locale sau de la distanta). Libraria GINA (Graphical Identification And Authentication) este incarcata in cadrul procesului winlogon, si ajuta la log-area utilizatorilor locali.

2.Mecansime de editare

Mecanismele de ediate sunt:

2.1 Legarea statica

Legarea statica a fost varinta initiala de a lega programele cu diferitele rutine din biblioteci neceasare acestora. Linker-ul primeste codul modului obiect cu mai multe referinte nerezolvate catre modulele obiect din diverse biblioteci.Linker-ul rezolva toate aceste referinte dupa care genereaza un singur fisier care contine atat codul programului cat si parti din mai multe biblioteci continand modulele necesare pentru functionarea programului. . Inainte de generarea fisierului executabil se va aloca memorie pentru referintele externe ale programului.

Avantajele acestui mecanism:

- modul simplu de implementare si la portabilitatea modulelor obtinute prin legarea statica
- timpii de incarcare a programelor legate static sunt mai mici fata de cele legarate dinamic

Cateva din dezavantajele acestui mecanism sunt:

- se pot irosi resurse RAM importante prin duplicarea aceluiasi cod dintr-o biblioteca pentru fiecare program ce foloseste legarea statica. Pot fi irosite deasemena resurse de stocare importante deoarece programele contin toate modulele de care ar putea vreodata sa aiba nevoie, astfel un program poate fi ridicol de mare.
- un program obtinut cu ajutorul legarii statice contine un subset de module obiect impartite in mai multe librarii. Acestea nu pot fi reunite pentru a pune impreuna modulele care se apeleaza reciproc, lucru ce ar imbunatati aplicatia.
- versiunile ulterioare ale sistemelor de operare au incluse biblioteci ce contin rutine optimizate ,dar programele legate static folosesc tot vechile rutine neoptimizate.

2.2 Legarea dinamica

Atunci cand editorul de legaturi foloseste legarea dinamica pentru construirea unei aplicatii vor fi rezolvate toate referintele catre modulele necesare dar acestea nu vor fi copiate in varianta executabila. Linkerul adauga etichete de start-up cu ajutorul carora vor fi incarcate librariile necesarae la rularea aplicatiiei. Fiecare apel al unei librarii este pastrat intr-un tabel unde se trece locatia respectivei librarii, astfle ca la urmatoarea apelare se va folosi referinta indirecta din tabel.

Este acea legare cu “viata scurta” ce se face imediat inaintea unei singure executii, precedata de o analiza a sistemului si a aplicatiei prin care se evidentiaza modulele necesare programului in acel moment.

Legarea dinamica desi pare mai laborioasa este mai utila la sistemele complexe formate din mai multe module care nu ar putea fi incarcate toate in memorie la un moment dat. Inainte de executie se va face un bilant al modulelor necesare si acestea vor fi legate si apoi incarcate in momentul executiei.

Legarea dinamica se face in doua faze:

- in faza I se verifica modulele si se completeaza ESTAB-ul
- in faza II se va face legarea propriu-zisa si se va completa cu valori externe

In organigramele ce descriu legarea dinamica vom folosi mai multe structuri:

- PROGADR – numarator cu adreasele de incarcare a programelor rulate
- CSADR – adresa de inceput pentru modulul care se incarca in momentul curent
- CSLTH – numarator cu lungimea modulului care se incarca in momentul curent
- ESTAB – tabel cu variabile externe si completat cu tabelul cu variabile publice, contine referirile comune din modulul complex
- EXECADR – va contine adresa de start a intregului program

Faza I:

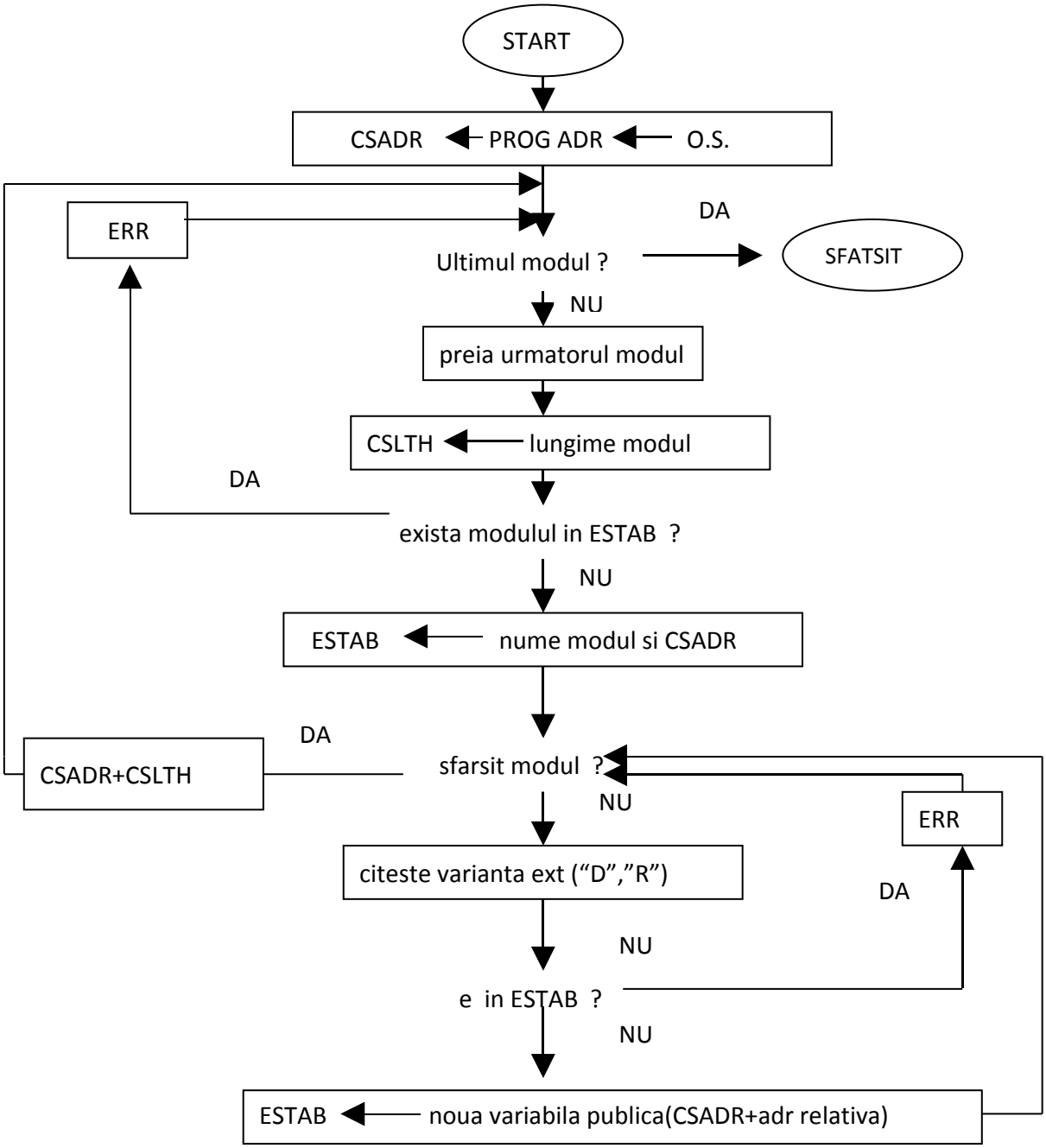
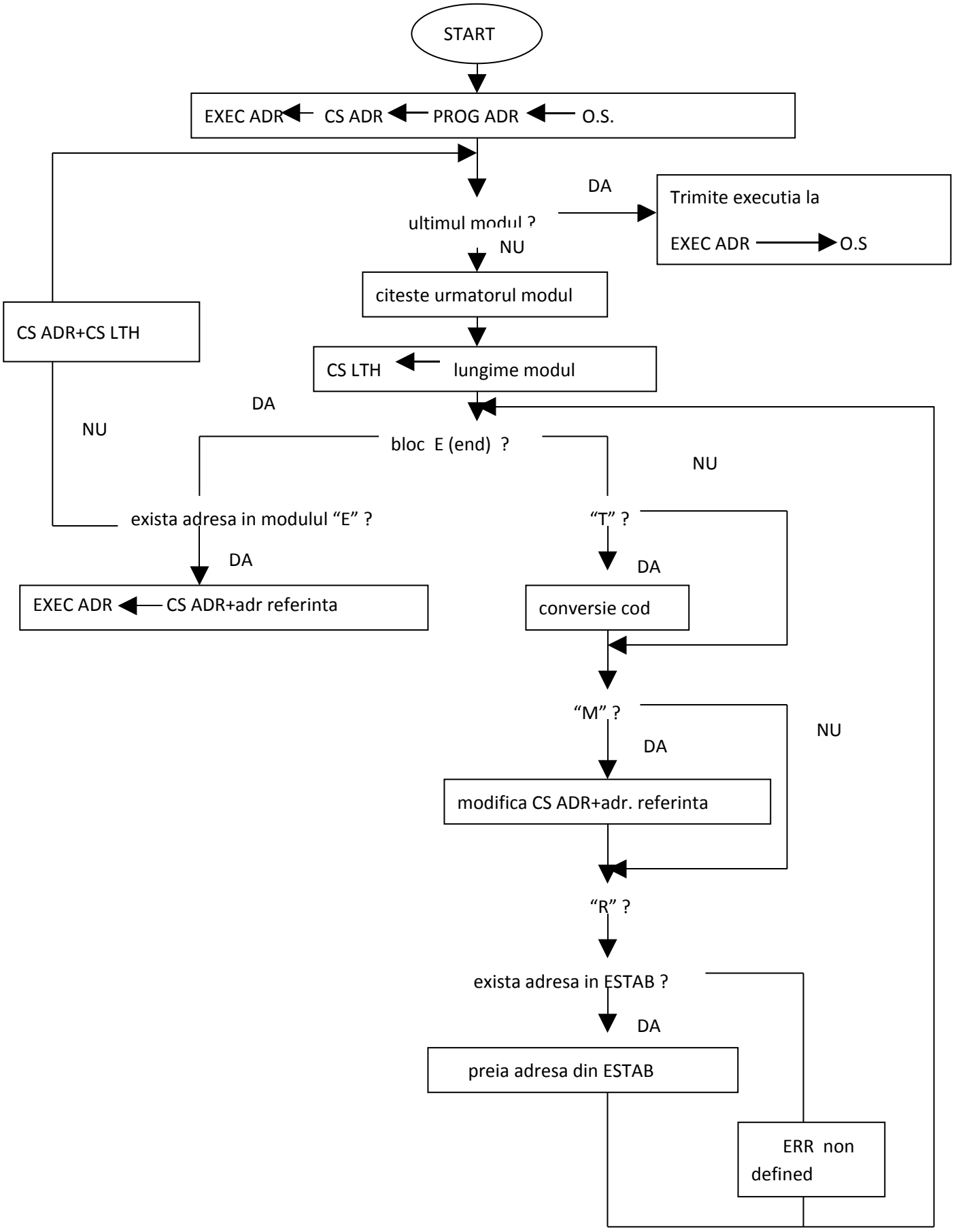


Fig . 1[1]

[1,2-] Note curs: Curs 5, subcapitolul „Legarea dinamica”

Faza a II-a:

Fig. 2^2



Avantajele legarii dinamice sunt:

- se reduce spatiul de stocare deoarece mai multe aplicatii vor folosoi mai multe instante ale aceleiasi librarii, fara sa fie nevoie sa o copieze fiecare in parte.
- impartirea unei aplicatii in mai multe module legate dinamic va ajuta la imbunatatirea securitatii, aplicatia fiind mai greu de dezasamblat.
- timpul de incarcare a unei aplicatii poate fi mai scazut daca modulele necesare sunt deja incarcate in memoria operationala.
- modulele nu sunt legate static de aplicatii ci se lega dinamic la incarcarea acesteia, cea ce permite aplicatiei sa mosteneasca schimbarile suferite de module fara a fi necesara recompilarea sau refacerea legaturii.

Dezavantaje legarii dinamice sunt:

- performantele unui program legat dinamic sunt mai mici, deoarece se pierde timp pentru referintele catre modulele din bibliotecile comune,pentru fiecare referinta se vor pierde opt cicli de procesare.
- programele legate dinamic trebuie sa aiba biblioteci compatibile, daca o biblioteca este modificata atunci programele vor trebui modificate pentru a reface compatibilitatea cu noua biblioteca. Daca o biblioteca necesara uni program este stearsa atunci programul nu va mai functiona.

3. Structura unui modul obiect

Datorita includerii, fisierele preprocesate contin deja declaratiile obiectelor importate. Apoi ele sunt compilate separat . Rezultatul acestei compilari este ceea ce se numeste modul obiect (object file). Borlandc genereaza fisiere cu extensia .obj, compilatoarele din UNIX cu extensia .o. Se obtine cite unul pentru fiecare modul C initial.

Un modul (fisier) obiect este un amestec de program compilat (cod masina) si variate informatii. Tot textul C al modulului a fost compilat in cod masina. Insa referintele la simboluri externe din alte module nu au putut fi satisfacute, pentru ca aceste module sunt compilate separat. De aceea modulul obiect contine o lista a simbolurilor nesatisfacute, indicand si locurile unde sunt folosite. In plus mai contine si o lista a simbolurilor exportate de modul.

De asemenea un modul obiect are o parte cu informatii numite de relocatare. Acestea sunt necesare pentru ca toate salturile din codul compilat sar la niste adrese care se vor schimba probabil atunci cand modulul obiect va fi pus cap la cap cu altele pentru a forma executabilul.

Structutra unui modul obiect este standardizata, trebuie sa contina trei parti : inceput,cuprins si incheiere :

Cod	Nr octeti	
I (introducere)	1	„i”
	2-9	nume program
	10-13	adresa de memorie unde se incaraca prima locatie din fisierul obiect
	14-15	
C (cuprins)	16	„C”
	17-18	adresa de inceput pt. paragraful curent
	19-20	lungime paragraf curent
	21-148	octeti de cod obiect pentru paragraful curent
		Se poate repeta de n ori pana la xxx
E (sfarsit)	xxx +1	„E”
	(xxx+2) – (xxx+3)	adresa primei instructiuni de executat
	(xxx+4) – (xxx+5)	„EOF” (end of file)

Formatul obiect poate sa aiba o structura extinsa de forma „I”,”C”,”M”,”E”,”T”. Uunde sectiunea M contine modificarile pentru incrementarile necesare programului si T contine informatiile necesare pentru aplicarea unei masti.

cod	Nr octeti	
M	1	„M”
	2-7	locatia unde se face modificare
T	1	„T”
	2-7	Locatia inceput cod
	8-9	lungime cod
	10-17	masca relocatare
	18-274	Cod

3.1 Formatul COM

O imagine de tip COM este cel mai simplu tip de modul executabil, nefiind un format in sine deoarece nu exista un header pe care sistemul de operare sa il citeasca si sa il interpreteze. Astfel dupa ce se incarcarea in memorie, un fisier COM are exact aceeasi structura ca si imaginea sa de pe hard disk si contine doar date si instructiuni executabile. Sistemul de operare MS DOS nu va face vreo verificare referitoare la corectitudinea imaginii, ci o va incarca pur si simplu la offsetul 0x0100 in cadrul primului segment de memorie disponibil.

Dupa incarcarea in memorie, un program de tip COM are structura urmatoare

Offset	Continut
0x0000	PSP
0x0100	Punct fix de start
…	Cod, date
0xFFFE	SP (stack pointer)

O imagine de tip COM nu necesita relocari de simboluri datorita dimensiunii maxime de 64Kb (un segment).

Dupa asamblarea si link editarea unui program de tip COM se obtine o imagine executabila COM cu o dimensiune de 42 bytes.

3.2 Formatul MZ

Cresterea complexitatii a determinat cresterea dimensiunilor programelor si astfel bariera de 64Kb pntru formatul COM a devenit insuficienta. O solutie la aceasta problema ar fi ca un program sa cuprinda mai multe segmente de memorie, dar in acest caz ar fi trebuit specifcata complet adresa de start ar trebui sa fie specificata deoarece nu este cunoscuta implicit.

Solutia la aceasta problema a fost adaugarea la inceputul imaginii a unei portiuni de date denumita header care contine toate informatiile necesare pentru ca sistemul de operare sa incarce corect imaginea in memorie si este generat in mod transparent de catre link editor. Structura header-ului MZ este prezentata in tabelul urmator :

Offset	Dim.	Continut	Semnificatie
0x0000	2	0x5A4D ("MZ")	Semnatura MZ, folosita pentru verificarea corectitudinii imaginii.
0x0002	2	wPartPage	Lungime fisier modulo 512.
0x0004	2	wPageCnt	Lungime fisier in pagini de 512 bytes, inclusiv header-ul.
0x0006	2	wReloCnt	Numar de elemente in tabela de relocare.
0x0008	2	wHdrSize	Dimensiune header, exprimata in paragrafe de 16 bytes.
0x000A	2	wMinAlloc	Necesar minim de memorie peste sfârsitul programului (in paragrafe).
0x000C	2	wMaxAlloc	Necesar maxim de memorie peste sfârsitul programului (in paragrafe).
0x000E	2	wInitSS	Valoarea initiala a segmentului de stiva (SS).
0x0010	2	wInitSP	Valoare initiala stack pointer (SP).
0x0012	2	wChkSum	Suma de control. Este suma cu semn schimbat a tuturor cuvintelor din fisier.
0x0014	2	wInitIP	Valoare initiala instruction pointer (IP).
0x0016	2	wInitCS	Valoarea initiala a segmentului de cod (CS).
0x0018	2	wTablOff	Adresa tabelei de relocare in cadrul fisierului.
0x001A	2	wOverlayNo	Indicator overlay. Este 0 pentru un executabil de sine statator.
?	?	alReloTbl	Tabela de relocare
?	?	abFiller	Bytes fara semnificatie pâna la limita de paragraf.

Header-ul este completat cu octeti fara semnificatie pâna la o limita de paragraf.

Procesul de incarcarea in memorie a unei imagini MZ se executa de catre functia EXEC (0x4B). Dupa asamblarea si link editarea unui program de tip MZ se obtine o imagine executabila de 572 bytes, fiind o diferenta foarte mare fata de versiunea COM in conditiile in care functionalitatea este aceeasi. Motivul este, evident, prezenta header-ului.

3.3 Formatul NE

Acesta a aparut odata cu prima versiune de Microsoft Windows ca o consecinta a modificarilor de arhitectura introduse de acesta, si anume:

-abilitatea de a rula mai multe programe aparent simultan (multitasking) ;

-modul in care aceste programe utilizau functionalitatea sistemului de operare ;

A fost introdusa de asemenea si interfata grafica oferita de sistem intr-un mod transparent pentru programator, detaliile de implementare a ferestrelor, meniurilor, etc. fiind ascunse. Intregul cod oferit de sistem era continut in câteva biblioteci dinamice pe care programul le incarca in memorie dupa cum era necesar. Astfel a aparut notiunea de cod reutilizabil (in cazul folosirii bibliotecilor statice, orice program care le utilizeaza contine automat si codul existent in ele determinând o dimensiune mai mare a imaginii).

Structura unei imagini NE este urmatoarea :

Offset	Continut
0x0000	Header MZ
0x0020	Rezervat
0x003C	Adresa header NE
0x0040	Stub MS DOS (program MS DOS de mici dimensiuni care afiseaza mesajul "This program must be run under Microsoft Windows" in momentul in care se incearca rularea programului sub sistemul de operare MS DOS)
...	...
	Header NE
	Tabela de segmente
	Tabela de resurse
	Tabela de nume rezidente
	Tabela de referinta module
	Tabela de importuri
	Tabela de intrare
	Tabela de nume non-rezidente
	Date segment 1
	Caracteristici segment 1
	...
	Date segment n
	Caracteristici segment n

Componentele imaginii NE sunt:

3.3.1 Header-ul NE

Se afla la adresa specificata de cuvântul de la offsetul 0x003C din cadrul header-ului MZ si contine atât informatii despre structura imaginii cât si diverse valori referitoare la link editor sau compilator.

Structura unui header NE este urmatoarea :

Offset	Semnificatie
0x0000	Semnatura NE (0x4E45)
0x0002	Numar versiune link editor
0x0003	Numar revizie link editor
0x0004	Adresa tabelei de intrare (relativa la inceputul header-ului NE)
0x0006	Dimensiune in bytes a tabelei de intrare
0x0008	Rezervat
0x000C	Câmp de biti care descrie continutul imaginii. Valorile pe care le poate lua acesta vor fi descrise ulterior
0x000E	Numarul segmentului care contine "automatic data". Are valoarea 0 daca pozitiile SINGLEDATA si MULTIPLEDATA din câmpul de biti de mai sus nu sunt setate
0x0010	Dimensiunea initiala (in bytes) a memoriei libere locale (local heap).
0x0012	Dimensiunea initiala (in bytes) a stivei
0x0014	Valoarea CS:IP
0x0018	Valoarea SS:SP
0x001C	Numarul de intrari in tabela de segmente
0x001E	Numarul de intrari in tabela de referinta module
0x0020	Numarul de intrari in tabela de nume non-rezidente
0x0022	Adresa tabelei de segmente (relativa la inceputul header-ului NE)

Offset	Semnificatie
0x0024	Adresa tabelei de resurse (relativa la inceputul header-ului NE)
0x0026	Adresa tabelei de nume rezidente (relativa la inceputul header-ului NE)
0x0028	Adresa tabelei de referinta module (relativa la inceputul header-ului NE)
0x002A	Adresa tabelei de importuri (relativa la inceputul header-ului NE)
0x002C	Adresa tabelei de nume non-rezidente (relativa la inceputul imaginii)
0x0030	Numarul de adrese de start relocabile
0x0032	Dimensiunea la care se aliniaza segmentul. De exemplu, daca aici se afla valoarea 4, segmentul se va alinia la o dimensiune de $2^4 = 512$ bytes
0x0034	Numarul de segmente care contin resurse
0x0036	Câmp de biti care semnifica sistemul de operare pentru care a fost compilata imaginea. Se va detalia ulterior
0x0037	Câmp de biti care contine informatii aditionale despre imagine. Se va detalia ulterior
0x0038	Rezervat
0x003A	Rezervat
0x003C	Rezervat
0x003E	Versiunea de Windows pe care imaginea se asteapta sa ruleze

3.3.2 Tabelel formatului NE

Tabela de segmente

Contine informatii despre toate segmentele prezente in imaginea executabila (dimensiune, tip, detalii de relocare) . In tabelul de mai jos este descris structura unui element din acest tabel, adresele sunt exprimate relativ la inceputul elementului.

Structura unei intrari in tabela de segmente este urmatoarea :

Offset	Descriere
0x0000	Adresa datelor continute in segment (relativ la inceputul imaginii). O valoare 0 semnifica absenta datelor
0x0002	Dimensiunea (in bytes) a segmentului. O valoare 0 semnifica o dimensiune de 64Kb
0x0004	Câmp de biti care descriu continutul segmentului. Se va detalia ulterior
0x0006	Valoarea minima (in bytes) a memoriei alocate pentru segment. O valoare 0 semnifica un minim de 64Kb

Tabela de resurse

Tabela descrie si identifica locatia fiecarei resurse existente in modulul executabil. Resursele sunt reprezentate de meniuri, casete de dialog, fonturi sau bitmap-uri, date pe care programul le acceseaza in timpul executiei sale.

Membrii acestei structuri (RSRC_Entry)au urmatoarea semnificatie:

rscAlignShift - Tipul de aliniere folosit pentru a stoca respectiva resursa

rscTypes - Tabela de structuri de tipul TYPEINFO. Fiecare intrare descrie detaliat tipul respectiv de resursa.

rscEndTypes - Marcheaza sfârsitul tabelei rscTypes. Are valoarea 0 intotdeauna

rscResourceNames - Tabela de nume asociate cu fiecare resursa. Primul byte din fiecare intrare specifica numarul de caractere pe care le are numele datorita faptului ca acesta nu se stocheaza sub forma unui sir terminat cu 0

rscEndNames - Marcheaza sfarsitul tabelei rscResourceNames (0 intotdeauna).

Tabela de nume rezidente

Este alcatuita din nume de functii pe care imaginea le exporta (in cazul in care este vorba de un DLL. Aceste siruri de caractere sunt stocate in memoria de sistem si raman aici pe tot parcursul executiei programului. Numele sunt "case sensitive" si nu sunt terminate cu 0.

Prima intrare din tabela de nume rezidente este chiar numele modulului.

Tabela de nume non-rezidente

Aceasta tabela contine denumiri de functii exportate de catre modul. Diferenta fata de 3.4 este ca tabela nu se afla in zona de memorie rezidenta, sistemului de operare avand posibilitatea de a utiliza aceasta memorie in alte scopuri. Prima intrare din tabela reprezinta un "description string" (sir de caractere definit de programator in procesul de link editare). Structura este identica cu cea a tabelei de nume rezidente.

Tabela de referinta module

Contine adresele numelor de module din care programul importa functii (nume care se regasesc in tabela de importuri). Fiecare intrare din tabela are o dimensiune de 2 bytes.

Tabela de importuri

Contine numele modulelor din care executabilul importa functii. Un element din tabela este organizat sub forma unui byte care specifica lungimea sirului, urmat de sirul propriu-zis (acesta nu este terminat cu 0).

Tabela de intrari

Este alcatuita din 2 blocuri, fiecare bloc având un header cu o dimensiune de 2 bytes. Header-ul respectiv are urmatoarea structura:

- primul byte semnifica numarul de intrari in blocul respectiv ; o valoare 0 semnaleaza sfârsitul blocului ;

- al doilea byte indica tipul segmentului in care se afla codul adresat de intrari (ex : o valoare de 0xFF semnifica atributul MOVABLE ; o valoare 0xFE indica faptul ca reprezinta de fapt o constanta definita in modul ; o valoare diferita de 0xFF sau 0xFE indica prezenta unui index in tabela de segmente)

3.4 Formatul PE

Trecerea de la sistemele de operare pe 16 biti (Windows 2.x, Windows 3.x) la cele pe 32 de biti, si anume Windows 95 si Windows NT a determinat aparitia acestui tip de format. S-a eliminat segmentarea memoriei, oferind unui program posibilitatea de a accesa pâna la 4Gb de memorie folosind un singur pointer ce semnifica un offset pe 32 de biti. Astfel modelul de memorie in care functioneaza o imagine PE a fost denumit FLAT (un singur "segment" cu o dimensiune de 4 Gb).

Numele de "portable executable" semnifica faptul ca o imagine compilata pentru Intel x86 va avea aceeasi structura cu una destinata microprocesorului Alpha (difera doar codificarea setului de instructiuni). Formatul PE nu constituie o noutate absoluta deoarece a imprumutat mult de la formatul COFF (Common Object File Format) care este caracteristic imaginilor executabile de pe platformele UNIX.

3.4.1 Structura unei imagini PE

Offset	Continut
0x0000	Header MZ
0x003C	Adresa header PE
0x0040	Stub MS DOS
	Header PE
	Tabela de sectiuni
	Sectiune 1
	Sectiune 2
	…
	Sectiune *n*

Se observa ca, la fel ca in cazul NE, o imagine PE contine un stub pentru executia in MS DOS. De asemenea, in header-ul acestui stub se gaseste un pointer catre header-ul PE.

3.4.2 Header-ul PE

Ca in cazul MZ sau NE, header-ul PE contine informatii despre structura imaginii si despre felul in care sistemul de operare va interpreta diverse portiuni ale acesteia cum ar fi adresele si dimensiunile sectiunilor, versiunea de sistem sau dimensiunea initiala a stivei.

Header-ul PE este descris in fisierul antet WINNT.H ca o structura de forma urmatoare :

```
typedef struct _IMAGE_NT_HEADERS
{
    DWORD                   Signature;
    IMAGE_FILE_HEADER       FileHeader;
    IMAGE_OPTIONAL_HEADER   OptionalHeader;
}IMAGE_NT_HEADERS, *PIMAGE_NT_HEADERS;
```

Membrii structurii IMAGE_NT_HEADERS au urmatoarele semnificatii:

Signature - Semnatura PE (0x50450000)

FileHeader - O structura de tipul IMAGE_FILE_HEADER

OptionalHeader - O structura de tipul IMAGE_OPTIONAL_HEADER

FileHeader contine informatii de interes general despre imagine si aceasta structura poate fi gasita si in cadrul fisierelor obiect produse de link editoarele pe 32 de biti.

OptionalHeader nu este deloc un membru optional, el continând multe date importante.

3.4.3 Tabelel formatului PE

Tabela de sectiuni

Tabela de sectiuni se afla intre header-ul PE si prima sectiune . Contine informatii despre toate sectiunile prezente in imagine (sectiunile sunt sortate in ordinea RVA si nu dupa nume).

Exista o analogie intre imaginile NE si imaginile PE.

O imagine NE grupeaza codul, datele si resursele in segmente care sunt descrise de diverse tabele imediat urmatoare header-ului. Deasemenea, informatii referitoare la segmente pot fi gasite in header precum si la sfârsitul fiecarui segment alaturi de adresele diverselor portiuni de date sau cod.

Astfel, dupa incarcarea in memorie, o imagine NE sa aiba o structura complet diferita fata de cea de pe hard disk.

In ceea ce priveste imaginile de tip PE arhitectura pe 32 de biti permite folosirea unor asa-numite "segmente" cu o dimensiune mai mare de 64Kb, astfel incât codul sau datele aplicatiei pot fi grupate fiecare in câte un bloc denumit sectiune de cod si, respectiv, de date. In acest fel, nu mai este nevoie de numarul mare de adrese pe care le continea tabela de segmente NE, fiind suficient sa precizam adresa la care incepe o anumita sectiune.

Se poate concluziona ca legatura dintre tabela de segmente si tabela de sectiuni o reprezinta faptul ca o sectiune ar corespundeunui segment adresabil pe 32 de biti.

Imagine PE functioneaza in modelul de memorie FLAT si astfel sectiunile pot fi definite ca fiind pur si simplu zone de memorie in cadrul spatiului virtual de memorie al imaginii. De asemenea, specificatiile PE garanteaza ca datele unei sectiuni se vor afla intr-o zona continua de memorie (la adrese consecutive).

O alta diferenta notabila o constituie faptul ca formatul NE nu stocheaza nici resursele, nici informatiile referitoare la functii importate (sau exportate) in segmente. La formatul PE, acestea se gasesc tot sub forma de sectiuni.

Tabela de sectiuni contine un numar de intrari egal cu valoarea lui *NumberOfSections* (gasit in membrul *FileHeader* al structurii IMAGE_NT_HEADERS)

Un exemplu de sectiuni frecvent intalnite ar fi

a) Sectiunea .text contine tot codul aplicatiei, indiferent de numarul de module obiect din care s-a generat imaginea. Organizarea codului in mai multe sectiuni nu mai are sens, deoarece nu mai exista limitarea de 64Kb impusa de modelul segmentat.

b) Sectiuneactiunea .data. In aceasta sectiune se grupeaza toate datele initializate ale programului, si anume variabile globale, variabile statice precum si toate sirurile de caractere. Variabilele de tip local nu ocupa spatiu in aceasta sectiune, ele fiind situate in stiva. Atributele tipice ale sectiunii sunt INITIALIZED_DATA, READ si WRITE.

c) Sectiunea .bss. Aici se vor gasi toate datele neinitializate si din aceasta cauza sectiunea .bss nu ocupa loc in imagine (RawDataOffset este 0). Link editoarele Borland nu emit aceasta sectiune, adunând dimensiunea sa la cea a sectiunii .data. Are ca atribut tipic UNINITIALIZED_DATA.

d)Sectiunea .idata. Aceasta este sectiunea de importuri si in ea se gasesc toate informatiile referitoare la functii importate si nume de biblioteci dinamice utilizate. Atributele pe care aceasta sectiune le are in mod implicit sunt INITIALIZED_DATA, READ si WRITE.

e)Sectiunea .edata. Aceasta sectiune este intâlnita doar pentru imagini PE de tip DLL si reprezinta tabela de exporturi. Atributele tipice ale sectiunii sunt INITIALIZED_DATA si READ. Sectiunea nu poate avea in nici un caz atributul DISCARDABLE deoarece o imagine oarecare trebuie sa aiba posibilitatea sa obtina in mod explicit adresa unei functii din acest DLL folosind functia Windows *GetProcAddress*.

Este important sa precizam ca, daca imaginea este incarcata la adresa specificata de link editor, nu vor exista nici un fel de relocari, aceasta sectiune fiind complet ignorata. Ea se poate chiar elimina pentru a obtine o imagine de dimensiuni mai reduse (operatiunea nu este totusi recomandata pentru un DLL).

O descriere mai detaliata a unei imagini PE este in figura urmatoare:

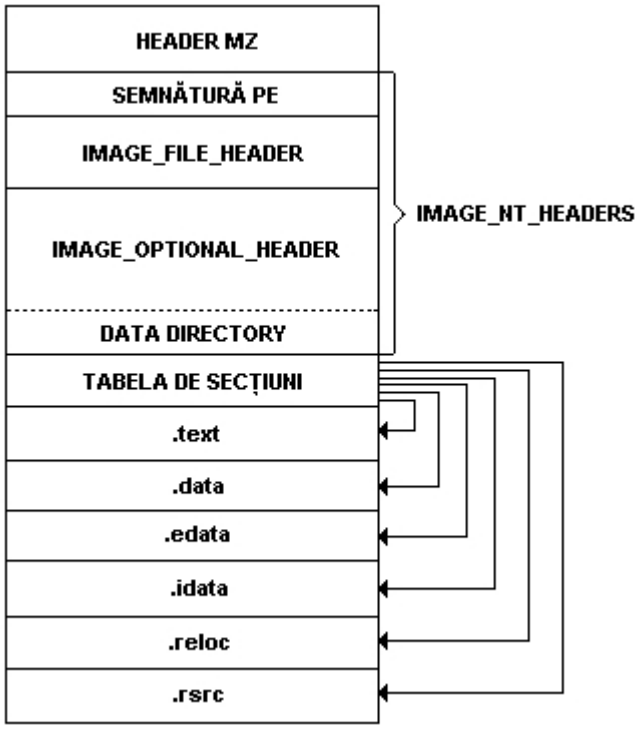


Tabela de importuri

Am vazut in cele de mai sus ca tabela de importuri se poate gasi intr-o sectiune proprie (.idata) sau in alte sectiuni, cum ar fi .rdata. Indiferent de locatie, aceasta tabela are intotdeauna aceeasi structura.

La inceputul tabelei se gaseste un vector de structuri de tipul IMAGE_IMPORT_DESCRIPTOR. Fiecare element al acestui vector poate fi descris de o structura cu urmatoarea forma :

```
typedef struct _IMAGE_IMPORT_DESCRIPTOR
{
    DWORD       OriginalFirstThunk;
    DWORD       TimeDateStamp;
    DWORD       ForwarderChain;
    DWORD       Name;
    DWORD       FirstThunk;
```

}IMAGE_IMPORT_DESCRIPTOR;

Membrii acestei structuri(IMAGE_IMPORT_DESCRIPTOR) au urmatoarele semnificatii:

OriginalFirstThunk - Este adresa unui vector de pointeri la structuri de tipul IMAGE_IMPORT_BY_NAME

TimeDateStamp - Indica data la care a fost link editata imaginea

ForwarderChain - Reprezinta un index in vectorul adresat de membrul FirstThunk. Functiile indicate de acest index sunt de tip "forward", adica exista in alt DLL decât cel importat. Functiile forward sunt folosite foarte rar si modul in care se face forwardingul nu este documentat.

Name - Reprezinta adresa unui sir de caractere terminat cu 0 care contine numele bibliotecii dinamice din care se importa functii.

FirstThunk - Este adresa unui vector de pointeri la structuri de tipul IMAGE_IMPORT_BY_NAME

In primul rând, trebuie mentionat ca, pentru fiecare DLL din care imaginea importa functii, exista câte o intrare de tipul IMAGE_IMPORT_DESCRIPTOR. Datorita faptului ca numarul de elemente din acest vector nu este specificat nicaieri, ultimul element este tot timpul completat cu 0 pentru a face posibila enumerarea.

Tabela de exporturi

Aceasta poate fi gasita in sectiunea .edata sau imediat dupa tabela de importuri in cadrul sectiunii .rdata. Este caracteristica bibliotecilor dinamice deoarece modulele executabile nu exporta in mod obisnuit simboluri.

Tabela de exporturi incepe cu o structura de tipul IMAGE_EXPORT_DIRECTORY care este definita in WINNT.H .

4.Legarea dinamica : comparatie Multix, Windows, Linux

Mecanismul de legare dinamica, “dynamic loader” (DL), poate fi inclus in sisitemul de operare, in programul cu care s-a scris aplicatia sau poate fi un program utilitar de sine statator. Dupa incarcare, DL transmite executia programului dupa legarea acestuia, preia rezultatul si reia operatia de legare, pana la incheierea prelucrarilor. Astfel utilizatorul are la dispozitie anumite module elementare, dar foarte variate, cu care formeaza segmente ce pot fi incarcate in memorie. [....]

4.1 Legarea dinamica sub Multics

Legarea dinamica presupune amanarea cautarii si maparii unei subrutine (sau a unui segment de date) pana la primul apel catre acea subrutina(sau folosire a segmentului de date). Pentru a realiza acest lucru, compilatorul adauga o segventa speciala de biti in codul obiect al uni program compilat, care, daca este adresat indirect produce o intrerupere, trimitandu-l catre linker-ul dinamic. Acesta analizeaza locatia care a cauzat intreruperea si cu ajutorul pointerilor gasit acolo afla numele simbolic al programului apelat(sau al segmentului de date referit). Apoi localizeaza segmentul respectiv, il mapeaza in spatiul curent de adrese si inlocuieste cuvantul indirect cu unul ce contine adresa programului, pentru ca viitoarele referinte sa nu mai produca intreruperi catre linker.

Exista mai multe feluri in care poate fi folosia legarea dinamica, urmatoarele fiind cele mai importante:

- pentru a permite depanarea initiala a unei colectii de programe, inainte ca aceasta sa fie codata complet.
- pentru ai permite unui program sa incuda un apel conditionat catre un alt program elaborat pentru managementul erorilor (sau alte programe specializate) , fara a fi nevoie de o cautare/mapare in prealabil a acestuia din urma, decat in momentul aparitiei erorii (sau a unei situatii speciale).
- pentru a permite unui grup de programatori sa lucreze la o colectie de programe interdependente, astfel incat fiecare sa aiba acces la cele mai noi subrutine de indata ce sunt disponibile.

Ori de cate ori mai multe programe « inrudite » sunt incarcate in memoria operationala, legaturile necesare vor fi facute de linker pentru fiecare program in parte in momentul executiei. Daca se stie ca un set de programe inrudite necesita anumite legaturi comune, atunci un program numit 'Binder' poate fi folosit pentru a le grupa intr-un singur segment, reunind astfel toate legaturile si referintele exterioare intr-o singura legatura cu exteriorul.

4.2 Legarea dinamica sub Linux

Pentru legarea dinamica linker-ul ELF(Exceutable and Linkable Format) se foloseste de doua tabele specifice procesorului :Tablou Global de Inceput (Global Offset Table) si Tablou de Legaturi a Procedurilor (Procedure Linkage Table) .

-Tablou Global de Inceput (Global Offset Table) :linkerele ELF suporta cod PIC (cod independent de pozitie) prin intermediul GOT in fiecare biblioteca comuna.GOT contine adresele absolute catre datele statica referite in program.Adresa GOT este in mod normal pastrata intr-un registru(EBX) care este o adresa relativa din codul care o refera.

-Tablou de Legaturi a Procedurilor (Procedure Linkage Table): redirectioneaza apelurile functiilor relocatabile catre adresele absolute.

In afara celor doua tablouri linkerul recurge si la: ".dynsym" care contine toate simbolurile importate si exportate ale fisierului,".dynstr" care contine siruri de nume pentru simboluri,".hash" care contine tabela hash pe care o poate folosi linkerul in timpul executiei pentru a cauta rapid simboluri si ".dynamic" care este o lista de pointeri si etichete.

Modul de functionare:

In Linux linkerul dinamic ld.so este el insusi o biblioteca comuna ELF. La rularea programului sistemul mapeaza ld.so intr-o zona a spatiului de adrese si ruleaza codul sau de inceput. Linkerul relocateaza si rezolva referintele catre rutinele sale care sunt necesare pentru a incarca tot ce este nevoie ulterior.

Linkerul dinamic creaza o lista pentru executabil care contine bibliotecile ce trebuie incarcate.Pntru fiecare intrare in lista linkerul cauta fiserul care contine biblioteca. Dupa ce a fost gasit fisierul , linkerul citeste headerul ELF pentru a gasi headerul programului care indica segmentul dinamic. Linkerul mapeaza biblioteca in spatial de adrese curent. Din segmentul dinamic adauga tabloul de simboluri al bibliotecii la seria de tablouri de simboluri si daca bibliotecile au la randul lor alte dependente adauga si acele biblioteci la lista care contine bibliotecile ce trebuie incarcate si procesul este continuat. Pentru clarificare,creeaza de fapt o harta structurata de linkuri pentru fiecare biblioteca si o adauga intr-o lista globala de legare. Linkerul memoreaza o lista de legatura a tablourilor in fiecare fisier.

4.3 Legarea dinamica sub Windows

Legarea dinamica folosita de Windows este similara cu cea din Linux, diferenta consta in faptul ca aici linker-ul face parte din kernel. In prima faza kernelul mapeaza executabilele sub ghidarea headerelor formatelor PE, apoi incarcatorul analizeaza IAT-ul (Import Address Table) modulului respectiv si vede daca dll-ul respectiv depinde de alte dll-uri si daca da le va mapa si pe acestea din urama. Procesul continua pana cand toate modulele dependente au fost mapate in memorie.

Cuprins:

3.4.3 Tabele ale formatului PE

- 4. Legarea dinamica : comparatie Multix, Windows, Linux
  - 4.1 Legarea dinamica sub Multics
  - 4.2 Legarea dinamica in Linux
  - 4.3 Legarea dinamica sub Windows

Bibliografie:

1.Activitatile realizate de editorul de legaturi **(Gulie Valentin)**

Stefan Stancescu SOFTWARE DE SISTEM - Note de curs

IBM Corporation, Operating System 360, Linkage Editor, Program Logic Manual, 1967

Russinovich Mark; David Solomon (2005). "Startup and Shutdown", *Microsoft Windows Internals*, 4th edition, Microsoft Press, pg. 251-273.

2.Mecansime de editare: **(Cone Serban)**

http://sunsite.uakom.sk/sunworldonline/swol-02-1996/swol-02-perf.html

http://www.cse.nd.edu/~dthain/courses/classconf/worts2006/WangLiu.ppt

http://publib.boulder.ibm.com/infocenter/systems/index.jsp?topic=/com.ibm.aix.prftungd/doc/prftungd/when_dyn_linking_static_linking.htm

Note curs: Curs 5, subcapitolul „Legarea dinamica”

3. Structura unui modul obiect **(Jderu Adrian)**

Software de sistem, Ştefan Stăncescu, note de curs, Universitatea Politehnică Bucureşti, Facultatea de Electronică şi Telecomunicaţii

4.Legarea dinamica : comparatie Multix, Windows, Linux (**Cone Serban**)

Elliott I. Organick, *The Multics System: An Examination of Its Structure* (MIT Press, 1972)

http://www.securityfocus.com/infocus/1872

http://www.securityfocus.com/infocus/1873

1 -