**Editarea legaturilor**

Tudor Alin

Vîrlan Simona

Chiroiu Emilia

**Cuprins:**

**1.Activitatile realizate de editorul de lagaturi (Tudor Alin)**

1.1 Incarcarea programelor relocabile

1.2 Relocarea memoriei

1.3 Incarcarea codului nerelocabil

1.4 Activitatile aditionale ale editorului de legaturi

**2. Mecansime de editare (Vîrlan Simona)**

2.1 Biblioteci statice si dinamice

2.2 Legarea statica

2.3 Legarea dinamica

**3. Structura unui modul obiect ( formatul COFF) (Chiroiu Emilia)**

3.1 Antet fisier (File Header)

3.2 Antet optional (Optional Header)

3.3 Antet sectiune (Section Header)

3.4 Tabel relocare sectiune (Section Relocation Table)

3.5 Tabel numar linie sectiune (Section Line Number Table)

3.6 Tabel simboluri (Table Symbol)

3.7 Tabel sir de caractere (Table String)

1. **Activitatile realizate de editorul de legaturi**

Operatiile de editare ajuta la crearea de programe complexe structurate. Un editor de legaturi primeste unul sau mai multe fisiere obiect generate cu ajutorul compilatorului, avand ca sarcina combinarea combinarea unor informatii din acestea pentru obtinerea unei imagini executabile valide mai exact programul executabil propriu zis sau obtinerea unui fisier librarie (DLL – Dynamic-Link Lybrary).

Obiectele sunt module de proram ce contin coduri masina si informatii necesare pentru conexiune. Aceasta vine in special sub forma unor simboluri (definitii symbol ) ce se impart in doua categorii :

* Simboluri definite sau exportate
* Simboluri nedefinite sau importate

Simbolurile definite sau exportate sunt variabile sau functii prezente in modul si sunt disponibile utilizarii spre alte module

Simbolurile nedefinite sunt variabile sau functii apelate de catre modulul prezentat ca obiect dar care nu sunt deifinite in interiorul acestuia.

Pentru UNIX termenul de “loader” sau incarcator este folosit ca sinonim pentru “linker”. Procesul de incarcare este reprezentat de trecerea in memoria operationala a programului asamblat si pregatit de executie. Exista cazuri cand incarcarea presupune implicarea alocarii unor zone de memorie si setarea unor biti de protectie iar alte cazuri cand doar se copiaza date pe disc in memoria operationala.

**1.1 Incarcarea programelor relocabile**

Pentru incarcarea programelor relocabile **sunt necesare urmatoarele structuri** :

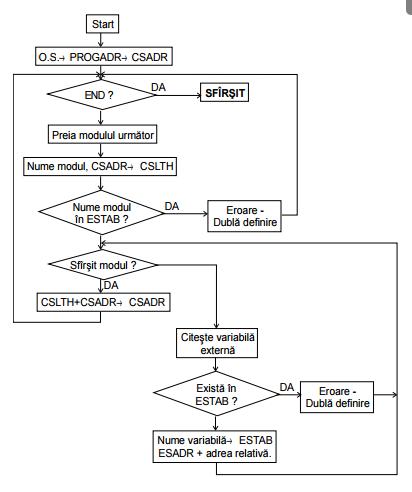
**ESTAB** –unde se gasesc variabilele externe cu nume si adresa si toate referintele **PUBLIC**

**CSADDR** –contine adresa de inceput pentru modulul current care se prelucreaza si in carese gaseseste referinta pentru relocarea adreselor relative din modul.

**CSLTH** –se gaseste lungimea modulului curent

**PROGADDR** –contine adresele de incarcare a programelor link-uite si adresa de start

**EXECADDR** –contine adresa de unde se va executa programul



**1.2 Relocarea memoriei**

Relocarea reprezinta proprietatea programelor de a fi executate in urma mutarii (incarcarii) la o alta adresa decat cea initial. In modulele relocabile adresele sunt calculate ca mutari fata de o adresa de referinta ( cea a instructiunii initiale). In programele relocabile se gasesc doua tipuri de adrese:

* adrese absolute
* adrese modificabile

Cand se incarca in memorie un modul relocabil se opereaza asupra adreselor modificabile: se aduna adresa primei locatii , modulului.

**Avantaje:**

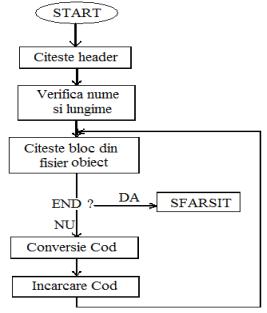
• posibilitatea de creare a unor programe mai complexe formate din mai multe module specializate.

• posibilitatea de rulare a unui program in diferite zone de memorie libere la momentul respective.

Editarea legaturilor este utila pentru constructia unor programme complexe prin alaturarea mai multe programme simple, facandu-le astfel mult mai compacte. Un program alcatuit din module relocabile este deasemenea relocabil, fiind incarat in system la orice adresa convenita intr-o zona de memorie libera.

**1.3 Incarcarea codului nerelocabil**

Incarcarea de cod nerelocabil (absolut ) muta codul obiect din fisier in codul obiect din memorie. Codul obiect trebuie sa contina un inceput , continut( cuprins ) si sfarsit. Procesul de incarcare preia parti de cod si le salveaza intr-o zona de memorie, acesta se face intr-un singur pas:



**1.4 Activitatile aditionale ale editorului de legaturi:**

* Inlocuirea subrutinelor in programul relocate o Include PLANNER (proglib)

o Delete PROJECT { DELETE from existing PLANNER } o Include PROJECT (NEWLIB) {INCLUDE new version } o Replace PLANNER ( PROGLIB)

* Construirea unui pachet de subrutine utlilizate in general, impreuna.

1. **Mecanisme de editare**

Inainte de a prezenta diferentele dintre legarea statica si cea dinamica, vom vedea întregul ciclu al unui program, de la scrierea codului sursa pana la executia acestuia. Programul este mai întai scris, utilizand un editor de text, apoi este compilat pentru a translate fisierul text în codul obiect pe care masina îl poate întelege si executa.

Programele scrise pot folosi alte programe sau biblioteci ale programelor. Acestea trebuie sa fie aduse împreuna cu programul pentru a îl putea executa.

Legarea este procesul de aducere impreuna a programelor externe. Legarea statica si legarea dinamica sunt doua procese de colectare si combinare a fisierelor obiect multiple, cu scopul de a crea un singur fisier executabil.

Legarea poate fi efectuata atat în timpul compilarii, atunci cand codul sursa este convertit în cod masina, cat si în timpul încarcarii programului în memorie, dar si în momentul rularii, prin programele de aplicatie. Legarea este efectuata prin programe numite linkeri, numiti si editori de legaturi. Legarea se face ca ultima etapa în elaborarea unui program.

Dupa legare, pentru executarea programului, acesta trebuie mutat în memorie. Pentru aceasta, trebuie safie alocate adrese datelor si instructiunilor în scopul executarii.

Procesul de mai sus poate fi rezumat ca ciclul de viata al unui program: scriere-compilare-legare-încarcare-executie.

**2.1 Biblioteci statice si dinamice [1]**

Cand un program in C este compilat, compilatorul genereaza codul obiect. Dupa generarea codului obiect, compilatorul cheama editorul de legaturi. Unul din principalele sarcini ale editorului de legaturi este sa faca codul functiilor pentru biblioteci (de exemplu, printf(), scanf(), sqrt() ) disponibile pentru program. Un editor de legaturi poate indeplini aceste sarcini copiind codul pentru functiile bibliotecilor in codul obiect, facandu-l disponibil in momentul rularii.

Din gruparea codurilor obiect ale fisierelor compilate intr-un singur fiser, cunoscut sub numele de biblioteca.

Exista doua tipuri de biblioteci Linux C/C++ ce pot fi create:

* Biblioteci statice (**.a**): biblioteci ale codului obiect cu care este legat; devine parte a aplicatiei.
* Biblioteci dinamice (**.so**): exista o singura forma a acestei biblioteci, dar poate fi utilizata in doua moduri:
  + Legate dinamic in momentul executiei. Bibliotecile trebuie sa fie disponibile in timpul fazei compilarii. Obiectele comune nu sunt incluse in fisierul executabil, dar au legatura cu executia.
  + Incarcate/descarcate dinamic si legate in timpul executarii utilizand functiile sistemului incarcator de legatura dinamica
  1. **Legarea statica [2]**

Legarea statica a fost prima metoda folosita pentru a combina un program cu parti din diferite rutine din biblioteci pe care acesta le utilizeaza. Este procesul de copiere a modulelor tuturor librariilor folosite în cadrul unui program în executabilul final.

Linker-ul combina codul de program cu rutinele din biblioteci pentru a rezolva referintele externe si pentru a genera un singur fisier potrivit pentru încarcarea în memorie, fisier ce contine codul programului si un amestec de module ce contin parti din mai multe biblioteci. Dupa ce programul este încarcat, sistemul de operare plaseaza în memorie fisierul executabil. Înainte de generarea acestui fisier se aloca memorie pentru referintele externe.

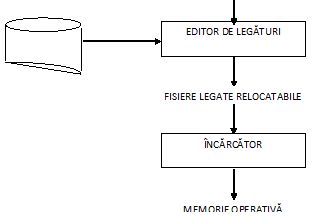


Fig.1 (Note curs: Curs 4.1, “Link-Load”)

* 1. **Legarea dinamica [2]**

In legarea dinamica, vor fi rezolvate toate referintele externe, dar nu vor fi copiate in fisierul executabil. Legatura efectiva etse realizata de sistemul de operare in momentul executiei, cand atat fisierul executabil, cat si bibliotecile sunt plasate in memorie. Legarea dinamica permite mai multor programe sa foloseasca o singura copie a unui fisier executabil. Acest lucru reduce semnificativ dimensiunea programului executabil, economisand astfel memorie si spatiu pe disc.

Editorul de legaturi adauga un cod de start-up pentru a încarca bibliotecile in timpul rularii, fiecare apel al unei biblioteci fiind memorat într-un tabel. Pentru apelurile ulterioare se folosesc referintele din tabel.  
 Legarea dinamica amana o mare parte din procesele de legare pana în momentul în care programul începe rularea.

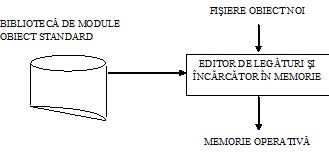


Fig.2 (Note curs: Curs 4.1, “Link-Load”)

Modelul legarii statice este simplu de implementat si de folosit, acesta fiind principalul avantaj al acestei metode. Insa, acest model este destul de limitat, astfel ca in momentul in care se face legarea unui program, fisierul executabil rezultat, contine toate datele si codul de care utilizatorul are nevoie. Schimbarea si extinderea comportamentului unui program implica recompilarea si refacerea procesului de legare. Utilizatorii sunt nevoiti sa închida aplicatia inainte de a instala o noua versiune a fisierului executabil de fiecare data cand are loc o actualizare a acestuia.

Cu legatura dinamica, programele pot partaja codul identic, în loc sadetina copii individuale ale aceleiasi biblioteci. De exemplu, avem bibliotecile standard ale limbajelor C sau C++. Fiecare program C sau C++ utilizeaza o parte din aceste biblioteci pentru I/O, alocarea dinamica, procesarea sirurilor si altele. Daca diferitele copii ale acestor biblioteci ar fi legate static de fiecare data în fiecare fisier executabil, chiar si programele cele mai mici ar ocupa zeci de megabytes. În plus, de fiecare data cand o noua versiune a bibliotecilor mentionate anterior este lansata, fiecare fisier executabil va trebui inlocuit cu altul nou pentru a reflecta schimbarea. Aceste biblioteci sunt, de obicei, implementate ca biblioteci dinamince încarcate în programul de baza în momentul rularii.

Avantaje legare dinamica:

* timpul de incarcare poate fi redus pentru ca biblioteca necesara se poate afla deja în memorie
* divizarea unei cereri în mai multe unitati de legatura face mai grea dezasamblarea fisierului executabil, ceea ce îmbunatateste securitatea
* rutinele nu sunt legate static de aplicatie, dar sunt legate dinamic în momentul în care aplicatia este încarcata. Acest lucru permite aplicatiilor sa mosteneasca automat modificari ale bibliotecilor partajate, f ar a recompilare sau refacerea legaturii

Dezavantaje legare dinamica:

* din punct de vedere al performantei, exista un “cod lipici”, care este necesar în programul executabil pentru a accesa segmental comun. Exista un cost de performanta în referintele pentru rutinele bibliotecilor distribuite, de aproximativ opt cicluri masina pe referinta. Programele care utilizeaza bibliotecile partajate sunt, de obicei mai lente decat cele care folosesc bibliotecile legate static

**De ce se foloseste legarea statica? [3]**

La compilarea unui program, în loc sa fie rescrise toate functiile, de fiecare data cand este scris un program nou, toate functiile de baza sunt pastrate in librarii. glibc este una dintre principalele biblioteci, ce contine codul tututor functiilor de baza utilizate de program (de exemplu, pentru deschiderea fisierelor, afisarea informatiilor pe ecran, interactiunea cu utilizatorul). Cand programul este compilat, aceste biblioteci sunt legate intre ele, impreuna cu noul program, astfel incat acesta programul sa poata utilize toate functiile pe care bibilioteca le are.

Totusi, aceste librarii pot fi foarte mari (de exemplu, libc.a are deseori 2.5MB), astfel ca nu mai pot fi separate copiile fiecarei librarii atasate programului.

Presupunem ca avem un fisier de 1KB si un alt fisier de 2.5MB, dar nu avem destul de mult spatiu (exceptand memoria RAM). Principalul avantaj al legarii dinamice este acela ca avem nevoie doar de o copie a bibliotecii. Daca instructiuniile ls sir m utilizeaza aceleasi bibioteci, atunci nu avem nevoie de doua copii ale bibliotecii, amandoua putand utiliza acelasi fisier. Asa ca, nu doar salvam spatial de pe hard disk, dar si memorie RAM.

Dar, desi legarea dinamica utilizeaza mult mai putina memorie, legarea statica este folosita mereu. Acest lucru se datoreaza faptului ca atunci cand directorul root pentru procesul current si pentru procesele derivate din acesta este schimbat intr-un mediu PFM chroot nou, aceste biblioteci dinamicenu vor fi disponibile pentru ca acestea vor fi in directorul vechi (de exemplu, /usr sau /lib), care nu sunt accesibile din chroot ales.

In aceasta situatie, pentru ca noile programe sa execute in noul mediu chroot, trebuie sa ne asiguram ca bibliotecile sunt legate static, pana cand sunt construite **(--enable-static-link, --disable-shared, -static**). Din acest motiv, este nevoie sa fie folosita legarea static.

**Exemplu pentru compararea performantelor legarii statice si legarii dinamice [4]:**

* **dimensiunea fisierului executabil (byte)**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Static | Dinamic (Incarcare) | Dinamic (executie) |
| Mic | 422596 | 5152 | 6300 |
| Mediu | 431413 | 5152 | 6297 |
| Mare | 443384 | 5134 | 6312 |

* **timpul de incarcare (us)**
* **timpul de executie (s)**
* **utilizarea memoriei (byte)**

1. **Structura unui modul obiect**
2. **Structura unui modul obiect**

Evolutia modulului obiect este strans legata de arhitectura de sistem de operare. Aparitia modelul de memorie FLAT, conceput ca un singur segment de dimensiune 4Gb si noua arhitectura de sisteme de operare pe 32 de biti dau nastere formatului Portable Executable, implementat incepand cu Windows NT. Acesta, precum si fisierele EXE (Executable), DLL(Dynamic-Link Library) au la baza formatul COFF. Echivalentul modulului obiect in cadrul Linux este ELF (Executable and Linkable Format).

Conform [5], se pot defini 2 forme generalizate ale modulului obiect (COFF) :

* fisier obiect ( object file/relocable file), rezultatul asamblarii si intrare(input) pentru editorul de legaturi. Acesta este relocabil, putand fi legat de alte module obiect sau combinat cu un fisier executabil dynamic pentru crearea imagine de proces;
* fisier executabil (image file), creat de catre editorul de legaturi din fisiere obiect, putand fi incarcat in memorie

Conform [6], componentele unui modul obiect, indiferent de stadiul in care se afla, se prezinta astfel:

|  |  |  |  |
| --- | --- | --- | --- |
| Structura | Functia | Locatia | Lungime |
| Antet fisier  (File Header) | Stocheaza informata de baza a fisierului si pointeri catre alte structuri. | La inceputul fisierului, exceptand fisierele executabile | Lungime de structura fixa. |
| Antet optional  (Optional Header) | Contine informatii aditionale despre executia fisierului. | Prezent dupa header-ul fisier, numai daca exista o indicatie in cadrul header-ul fisier. | Lungime specificata in fisierul header. |
| Antet sectiune  (Section Header) | Detine informatii despre sectiunile definite in fisier. | Succede header-ul optional iar in cazul absentei acestuia, header-ul fisier. | Produsul dintre numarului de sectiuni multiplicat si lungimea fixa de structura . |
| Tabel relocare sectiune  (Section Relocation Table) | Permite fisierului sa fie relocate catre orice zona de memorie, avand informatii despre adresele ce trebuie schimbate. | Exista cate un tabel pentru fiecare sectiune.  Locatia este indicata de catre antentul sectiune corespunzator. | Produsul dintre introducerile de relocari si lungimea fixa de structura. |
| Tabel numar linie sectiune  (Section Line Number Table) | Contine informatii de depanare pentru a mapa adrese de cod la liniile de cod din fisierul sursa . | Exista cate un tabel pentru fiecare sectiune.  Locatia este indicata de catre antentul sectiune corespunzator. | Produsul dintre numarul liniilor de cod si lungimea fixa de structura. |
| Tabel simboluri  (Table Symbol) | Stocheaza informatii despre fiecare symbol definit si declarat de catre cod. | Locatia este indicate de catre un pointer din Antetul fisier. | Produsul dintre numarul simbolurilor si lungimea fixa de structura. |
| Tabel sir de caractere  (Table String) | Contine nume de sectiune sau symbol cu o lungime mai mare de 8 caractere. | Succede Tabelul simboluri. | Lungime indicata de primii 32 de biti ai tabelului sir de caractere. |

* 1. **Antet fisier (File Header)**

Structura de inceput a formatului COFF are 20 Bytes, se gaseste in filehdr.h si contine:

{

**unsigned short f\_magic;** /\* Magic number \*/ - Conform [7], numarul variaza in functie de implementare. DJGPP genereaza fisiere COFF cu valoarea de 0X14 in acest spatiu. Este folosit pentru identificarea fisierului ca format.

**unsigned short f\_nscns;** /\* Number of Sections \*/ - Numarul de sectiuni indica cate structuri sectiune se gasesc in Tabelul sectiune

**long f\_timdat;** /\* Time & date stamp \*/ - Data si ora la care a fost creat fisierul

**long f\_symptr;** /\* File pointer to Symbol Table \*/ -Pointerul Symbol Table indica offsetul din fisier unde incepe Tabelul simboluri

**long f\_nsyms;** /\* Number of Symbols \*/ -Numarul de simboluri din Tabelul simboluri. Permite calculul zonei de incepere a Tabelului sir de caractere

**unsigned short f\_opthdr;** /\* sizeof(Optional Header)\*/ -Marimea Antetului optional

**unsigned short f\_flags;** /\* Flags \*/ -Flag-ul indica starea fisierului. Este folosit pentru o analiza rapida a informatiei ce poate fi calculata din restul fisierului. De exemplu, un bit indica daca informatia de relocare a fost scoasa din fisier, altul daca simbolurile externe au fost procesate.

}

* 1. **Antet optional (Optional Header)**

Header-ul optional cuprinde informatii despre rularea unui fisier, present numai in fisierele COFF executabile. Valoarea lui **f\_opthdr** diferita de zero marcheaza prezenta acestui Header.

Structura Optional Header are 28 Bytes, se gaseste in aouthdr.h si contine:

{ **unsigned short magic;**  /\* Magic Number \*/ - Contine numarul magic ce defineste formatul fisierului.

**unsigned short vstamp;** /\* Version stamp \*/ - Contine versiunea fisierului.

**unsigned long tsize;** /\* Text size in bytes \*/ - Contine marimea textului din fisier in bytes.

**unsigned long dsize;** /\* Initialised data size \*/ - Contine marimea datelor initializate.

**unsigned long bsize;** /\* Uninitialised data size\*/ - Contine marimea datelor neinitializate.

**unsigned long entry;** /\* Entry point\*/ Contine valoarea lui eip (pointerul de instructiune), inainte ca programul sa fie initializat.

**unsigned long text\_start;** /\* Base of Text used for this file \*/ Contine punctual de start al textului folosit pentru acest fisier.

**unsigned long data\_start;**  /\* Base of Data used for this file \*/ Contine puctul de start al datelor folosite pentru acest fisier.

}

* 1. **Antent sectiune (Section Header)**

Tabelul sectiune contine informatii despre sectiunile individuale continute in fisier. Acest table este un sir de structuri Antent sectiune . Lungimea acestuia este data de **f\_nscns**. Numerotarea in cazul acestui tabel se face incepand cu 1, deoarece sectiunea cu numarul 0 este rezervata.

Structura antetului are 40 Bytes, se gaseste in scnhdr.h si contine:

{ **char s\_name[8]**; /\* Section Name \*/ - Contine numele sectiunii, cu conditia sa nu depaseasca 8 caractere. Alftel, contine un pointer catre Tabel-ul sir de caractere.

**long s\_paddr;** /\* Physical Address \*/ - Contine adresa de start din memorie a fisierului. Daca acesta a fost legat de catre link-editor si urmeaza sa fie incarcat in memorie, aceasta variabila contine valoarea adresei din memorie.

**long s\_vaddr;** /\* Virtual Address \*/ - Aceasta este echivalentul adresei fizice, folosita pentru spatiul de memorie virtuala. In general, contine aceeasi valoare ca s\_paddr

**long s\_size;** /\* Section Size in Bytes \*/ - Contine marimea sectiunii in bytes.

**long s\_scnptr;** /\* File offset to the Section data \*/ - Contine offset-ul ce specifica inceputul zonei de date a acestei sectiuni. Poate fi folosit impreuna cu s\_size pentru a incarca sectiunea de date.

**long**  **s\_relptr;** /\* File offset to the Relocation table for this Section \*/ - Indica offset-ul fisierului pentru tabelul de relocare al sectiunii.

**long s\_lnnoptr;** /\* File offset to the Line Number table for this Section \*/ - Indica offset-ul fisierului pentru tabelul cu evidenta liniilor de date din sectiune.

**unsigned short s\_nreloc;** /\* Number of Relocation table entries \*/ - Numarul de inregistrari din tabelul de relocare.

**unsigned short s\_nlnno;**  /\* Number of Line Number table entries \*/ - Numarul de inregistrari din tabelul cu evidenta liniilor de cod. Daca este zero, atunci sectiunea nu contine informatii despre numarul liniei de cod

**long s\_flags;**  /\* Flags for this section \*/ - Aceasta variabila detine informatii inportante despre felul in care sectiunea trebuie tratata in timpul procesarii.

}

Cele mai importante valori ale s\_flags sunt urmatoarele:

0x0020: STYP\_TEXT \*/Sectiunea contine cod executabil. \*/

0x0040: STYP\_DATA \*/Sectiunea contine date initializate. \*/

0x0080: STYP\_BSS \*/Fisierul COFF nu contine date pentru aceasta sectiune, dar are nevoie de spatiu alocat. \*/

**3.4 Tabel relocare sectiune (Section Relocation Table)**

Tabelul de relocare este un sir de structuri cu inregistrari de relocare. Un fisier COFF poate contine un tabel de relocare pentru fiecare sectiune. Inregistrarile din Tabelul de relocare sunt folosite pentru a incarca o sectiunne de cod la o adresa de memorie arbitrara si a modifica liniile de cod pentru adresele de memorie a simbolurilor sa fie corecte. Aceasta actiune permite sectiunilor .text, .data sau .bss (Block Started by Symbol) sa fie incarcate la adrese de memorie diferite, iar sectiunea .text sa fie modificata in asa fel incat sa gaseasca valorile si functiile.

Tabelul de relocare pentru o sectiune isi are locatia si marimea stocate in antetul sectiune (Section Header)

Structura acestui antet are marimea de 10 Bytes, se afla in reloc.h, si contine:

{ **long r\_vaddr;** /\* Reference Address \*/ - Contine adresa de referinta

**long r\_symndx;** /\* Symbol index \*/ - Contine indexul de simbol

**unsigned short r\_type;** /\* Type of relocation\*/ - Contine tipul de relocare

}

**3.5** **Tabel numar linie sectiune (Section line number table)**

Un fisier COFF poate contine un tabel numar linie pentru fiecare sectiune, dar pentru sectiuni diferite de .text exista posibilitatea existentei informatiei de relocare. Inregistrarile de numar de linie sunt folosite de programele de depanare pentru a conecta simboluri si adrese fizice de instructiune la numarul liniilor din fisierul sursa.

La fel ca tabelul de relocare, tabelul numar linie pentru o sectiune are locatia si lungimea stocate in header-ul sectiune.

Structura acestui antent are lungimea de 6 Bytes, locatia sa este linenum.h si contine:

**{ union**

**{ long l\_symndx;** /\* Symbol Index \*/ -  **long l\_paddr;** /\* Physical Address \*/

**} l\_addr;**

**unsigned short l\_lnno;** /\* Line Number \*/

}

Tabelul contine o inregistrare cu numarul liniei de valoare 0 in care variabila l\_symndx va indica numele functiei simbol din Tabelul simboluri. Se incrementeaza si alte inregistrari care indica, prin l\_paddr, offset-ul sectiunii de unde incepe linia de cod. Astfel, o exceptie care apare in timpul procesarii de fisiere COFF poate fi urmarita pana la numarul liniei dintr-o functie.

**3.6** Tabel simbol ( Symbol table)

Tabelul simbol contine informatii despre fiecare simbol declarat sau definit in fisier. In acest tabel va exista cate un simbol pentru fiecare nume de functie, variabila globala, variabila statica sau externa. Exista si simboluri care indica inceputul fiecarei sectiuni si informatii aditionale despre fisier ca intreg.

Tabelul simbol este un sir cu structuri de inregistrari simbol. Locatia tabelului este continuta in f\_symptr din antetul fisier si lungimea data de f\_nsyms.

Structura antetului este in syms.h, are lungimea de 18 Bytes si contine:

{  **char n\_name[8];** /\* Symbol Name \*/ - Contine numele simbolului daca acesta nu depaseste 8 caractere, alftel contine un pointer catre tabelul sir de caractere

**long n\_value;** /\* Value of Symbol \*/ -Contine valoare numerica a simbolului

**short n\_scnum;** /\* Section Number \*/ - Contine sectiunea in care este continut simbolul. Daca valoare este mai mare ca zero, se refera la o sectiune din tabelul sectiune.

**unsigned short n\_type;** /\* Symbol Type \*/ - Contine tipul de simbol

**char n\_sclass;** /\* Storage Class \*/ - Contine clasa in care este este stocat simbolul.

**char n\_numaux;** /\* Auxiliary Count \*/ - Contine numerotarea auxiliara.

}

Variabila n\_scnum poate lua una din valorile predefinite:

N\_DEBUG (2) Un simbol de depanare.

N\_ABS (1) Un simbol absolut. Asta inseamna ca n\_value are o valoare absoluta.

N\_UNDEF (0) Un simbol extern nedefinit.

**3.7 Tabel sir de caractere (String Table)**

Tebelul sir de caractere contine nume de sectiuni si simboluri ce depasesc 8 caractere. Locatia sa se calculeaza folosind formula:

String Table Offset = File Header.f\_symptr + File Header.f\_symptr \* sizeof( Symbol Table Entry )

= File Header.f\_symptr + File Header.f\_symptr \* 18

Primii patru bytes ai tabelului sir de caractere sunt un intreg ce indica marimea globala in bytes a tabelului sir de caractere. Fiecare sir de caractere din tabel se termina in zero.

Prin prisma studiului numelor de simboluri si sectiuni, pentru a fi incluse in tabelul sir de caractere, se poate observa urmatoarea secventa:

{ union /\* Creaza o legatura intre numele simbolului, locstia si marimea sa.

{ char name[8] \*/

struct /\* Reprezinta o structura definita de variabila zeroea si offset \*/

{ unsigned long zeroes

unsigned long offset; }}}

Daca numele simbolului sau sectiunii are 8 caractere sau mai putin, variabila zeroes va fi diferita de zero si numele va fi interpretat ca un sir de caractere. Daca numele este mai lung de 8 caractere, variabila zeroes, reprezentand primii 4 biti ai variabilei name va fi zero. In acest caz, variabila offset este folosita pentru a stoca o valoare de offset in tabelul sir de caractere pentru locatia numelui simbolului sau sectiunii.

Structura unui modul obiect a evoluat, simplificandu-se odata cu introducerea conceptului de memorie FLAT, tratat ca un singur segment de 4 Gb. Astfel, modul de structurare al modulului obiect se axeaza pe descrierea exacta si tabelarea tuturor structurilor de baza care fac parte din morfologia unui limbaj de programare. Fiecare zona de cod este organizata in sectiuni iar fiecare sectiune contine simboluri si structuri bine definite in tabele specifice fiecarui tip de date. Astfel, editorul de legaturi are informatii exacte despre locatiile fiecarei parti din program si posibilitatea de legare in functie de nevoile utilizatorului, recombinarea si relocarea partilor de program dintr-un modul obiect efectuandu-se intr-un timp optim. Deasemenea, modulul obiect contine si informatii de depanare pentru a comunica eventualele erori sau exceptii in cadrul liniilor de cod.

**Bibliografie:**

[1] - Static, Shared Dynamic and Loadable Linux Libraries <http://www.yolinux.com/TUTORIALS/LibraryArchives-StaticAndDynamic.html>

[2] – Krishan Kumar , What are the differences between static and dynamic (shared) library linking?

<http://cs-fundamentals.com/tech-interview/c/difference-between-static-and-dynamic-linking.php>

[3] - Gerard Beekmans, Linux From Scratch: Chapter 5. Preparing the LFS system [http://oss.sgi.com/LDP/LDP/lfs/html/chapter05/whystatic.html#](http://oss.sgi.com/LDP/LDP/lfs/html/chapter05/whystatic.html)

[4] - Haitao Wang, Xiaomin Liu, Static Linking VS. Dynamic Linking, December 13, 2006 <https://www.google.ro/url?sa=t&rct=j&q=&esrc=s&source=web&cd=8&cad=rja&uact=8&ved=0CFgQFjAH&url=http%3A%2F%2Fwww3.nd.edu%2F~dthain%2Fcourses%2Fclassconf%2Fworts2006%2FWangLiu.ppt&ei=4YFjVfCoJY6R7Aax4oJ4&usg=AFQjCNHJpaWhMgpt7OusGNeU53tbiuRwUg&sig2=7An265hw9XaGuVl_QQBAfg&bvm=bv.93990622,d.ZGU>

[5] Sun Microsystems Inc. , Linker and Libraries Guide, Chapter 7, Object File Format

<http://docs.oracle.com/cd/E19683-01/817-3677/6mj8mbtc7/index.html>

[6] Microsoft PE/COFF

<http://wiki.osdev.org/COFF>

[7] DJ Delorie , DJGPP COFF Spec <http://www.delorie.com/djgpp/doc/coff/>

[8] Andrew S. Tanenbaum , Modern Operating Systems (4th Edition)

[9] Note de curs <http://stst.elia.pub.ro/programa/soIII.html>

[10] <http://osr507doc.sco.com/en/tools/ccs_linkedit.html>

[11]<http://upm.ro/intranet/ecalin/cd_educational/cd/s_o/sist_op.htm>l

[12] <http://www.webopedia.com/TERM/L/linker.htm>l

[13] <http://solomon.ipv6.club.tw/Course/SP.941/sp3-4.pd>

[14]<http://stst.elia.pub.ro/news/SOIII2009/Adrian_Petrila_Diploma_2003.doc>

[15]<http://stst.elia.pub.ro/news/SO/OS_PPT/stst_SO_C4.1_Link_LOAD.ppt>