EDITORUL DE LEGATURI

Parte tratata de Mihai Raciala(441A):

In domeniul IT , un conector sau un editor de legaturi este un program care ia unul sau mai multe obiecte generate de compilator si le asambleaza intr-un singur program executabil.
In cadrul interfetelor principale IBM ca OS/360 acest program este cunoscut ca un editor de legaturi.

(In variantele Unix termenul de "incarcare " este adeseori folosit ca sinonim pentru "legatura". . Pentru ca aceasta utilizare ingreuneaza distingerea intre procesul de compilare si cel de rulare , acest articol va folosi "legatura" pentru primul si "incarcare" pentru al doilea.. Oricum in unele sisteme de operare acelasi program administreaza ambele procese de conectare si incarcare a programului (vezi conectarea dinamica) .

Obiectele sunt module de program care contin coduri masina si informatii pentru conexiune. Aceasta informatie vine in special sub forma unor definitii simbol, care sunt de 2 feluri:

1simboluri definite sau exportate : sunt functii sau variabile care sunt prezente in modulul prezentat de obiect si care ar trebui sa fie disponibile utilizarii de catre alte module.

2Simboluri nedefinite sau importate sunt functii sau variabile care sunt chemate sau indicate de catre obiect dar nu sunt definite intern.

Pe scurt sarcina conectorului este de a rezolva referirile la simbolurile nedefinite prin gasirea altor obiecte care sa defineasca simbolul pus in discutie si sa inlocuiasca adresa acestuia cu adresa simbolului.

Editorii de legaturi pot lua obiectele dintr-o colectie numita biblioteca. Unele conexiuni nu includ toata biblioteca la iesire; ele includ doar acele simboluri care au ca referinta alte fisiere obiect sau biblioteci. Bibliotecile pentru diverse scopuri exista , si una sau mai multe biblioteci de system sunt de obicei conectate din oficiu.

Editorul de legaturi se ocupa de asemenea de aranjarea obiectelor intr-un spatiu de adresa pentru program . Aceasta ar putea include reasezarea codului care implica o adresa de baza specifica unei alte baze. Atata timp cat un compilator rareori stie unde se va gasi un obiect el isi asuma deseori o baza fixa de locatie (de exemplu 0). Reasezarea codului masina poate implica redirectionarea salturilor absolute , incarcarilor si depozitarilor.

Iesirea executata de editorul de legaturi ar putea avea nevoie de o relocalizare cand este in sfarsit incarcata in memorie (chiar inainte de executie). Pe memoria virtuala oferita de hardware aceasta este de obicei omisa , desi fiecare program este pus in spatiul propriei adrese , deci nu va exista nici un conflict chiar daca toate programele incarca aceeasi adresa de baza.

Legaturi dinamice

Mediile sistemelor de operare moderne permit utilizarea legaturilor dinamice, care reprezinta amanarea rezolvarii unui simbol nedefinit pana cand programul este folosit. Asta inseamna ca executabilul inca contine simboluri nedefinite , si inca o lista de obiecte sau bilioteci care vor furniza definitii despre acestea . Incarcarea programului va incarca de asemenea si aceste

biblioteci/ obiecte si va executa conexiunea finala.

Acest concept ofera doua avantaje :

2Bibliotecile deseori folosite (de exemplu bibliotecile standard ale sistemului) trebuie sa fie inmagazinate intr-o singura locatie , nu duplicate in fiecare binar.

3Daca o eroarea unuei functii din biblioteca este corectata prin inlocuirea bibliotecii , toate programele ce o utilizeaza vor beneficia de corectarea ei dupa restartarea lor. Programele care include aceasta functie prin conexiune/conectare statica vor trebui reconectate mai intai.

Relaxarea

Cum compiler-ul nu are nici o informatie despre aranjarea obiectelor din iesirea finala , nu poate profita de instructiunile mai scurte sau mai eficiente care plaseaza o cerere pe adresa altui obiect . De exemplu o instruciune "jump" poate indica o adresa absoluta sau un ofset din locatia curenta , si ofsetul ar putea fi exprimat prin diferite marimi in functie de distanta pana la tinta. Prin generarea celei mai conservative instructiuni (de obicei varianta absoluta , in functie de platforma) si adaugand indiciile de relaxare , este posibila substituirea instructiunii mai scurte sau mai eficiente pe durata legaturii finale. Acest pas poate fi facut doar dupa ce toate obiectele intrarii au fost citite si le-au fost alocate adrese temporare . Situatia de relaxare realoca ulterior adrese , ceea ce poate permite pe rand mai multe relaxari. In general secventele substituite sunt mai scurte , ceea ce permite procesului sa se concentreze intotdeauna asupra solutiei optime data intr-o aranjare fixa a obiectelor ; daca nu este cazul , relaxarile pot fi in conflict , si conectorul trebuie sa cantareasca avantajele fiecarei optiuni.

Neajunsurile editorului de legaturi

Editorul de legaturi al Unix-ului opereaza in mod traditional intr-un singur timp. Efectul acestui tip de functionare este acela ca functiile din librarii sunt incluse in imagine doar daca sunt necesare la timpul scanarii librariei. Acest lucru, combinat cu simboluri slabe si definitii multiple ale aceleiasi functii poate duce la efecte ciudate si neasteptate.

Consideram ca exemplu implementarea unei MPI (Message Passing Interface – program care permite mai multor calculatoare sa comunice intre ele) in care asocierea Fortran este realizata prin folosirea de functii wrapper peste implementarea in C. Autorul acestei librarii considera ca este rezonabil sa creezi functii specializate numai pentru legaturile C, in timp ce Fortran, eventual va apela aceste functii si costul wrapper-elor este presupus a fi mic. Totusi, daca functiile de tip wrapper nu se gasesc in libraria de profil, atunci nici o parte din codul profilului nu va fi nedefinita atunci cand va fi apelata libraria de profil. De aceea, nici o parte a codului profilului nu va fi inclusa in imagine. Cand libraria MPI standard este citita, functiile Fortran wrappers vor fi rezolvate si de asemenea se vor extrage variante de baza ale functiilor MPI. Efectul total este legarea cu success a codului dar nu va fi profilat.

Pentru a preveni acest lucru trebuie sa ne asiguram ca functiile wrapper din fortran sunt incluse in varianta profilata a librariei. Ne asiguram ca acest lucru este posibil, conditionand ca acestea sa fie separabile de restul librariei MPI principala. Acest lucru le permite sa fie extrase in afara librariei de baza si plasate intr-o librarie de profil folosind comanda Unix "ar".

Parte tratata de Lavinia Hiliti(441A):

Editorul de legături combină mai multe module obiect (.obj) diferite într-un singur modul absolut (.m66) care poate fi încărcat de un depanator sau emulator. El rezolvă referinţele publice şi externe şi părţile de programare locatabile sunt asignate la adrese absolute. De asemenea, editorul alege librăriile necesare şi leagă codurile rutinelor din librării la programul final. În final, editorul de legături produce un fişier obiect absolut care conţine întregul program, eventual şi informaţiile necesare pentru depanare.

Programele editoare de legături transformă programele din format obiect în programe executabile, realizând, dacă este cazul, integrarea mai multor module obiect într-un singur program executabil .

Editorul de legături grupează mai multe module obiect (rezultate în urma compilării sau preluate din biblioteci de module obiect) şi generează segmentele programului executabil.
Segmentul reprezintă o colecţie ordonată de secţiuni, între care au fost rezolvate legăturile definite prin elementele de comunicaţie. Un segment se caracterizează prin nume, o adresă de intrare în segment, adică adresa primei instrucţiuni executabile a segmentului, şi modulele obiect care îl alcătuiesc.

Pentru realizarea unui segment, editorul de legături îndeplineşte funcţiile următoare:

o realizează definirea completă a dicţionarului de legături a fiecărei secţiuni : pentru fiecare simbol extern apelat, verifică existenţa unei intrări într-un dicţionar de legături, şi completează dicţionarul de legături cu adresa acestuia ;

o alocă o zonă contiguă de memorie pentru segment, prin alocarea de locaţii succesive de memorie tuturor secţiunilor care îl alcătuiesc; pe baza acestei alocări se determină adresele de încărcare a secţiunilor în memorie, relativ la adresa 0 de încărcare a segmentului ;

o relocatează adresele ataşate simbolurilor externe, prin adunarea la aceste adrese a adresei de încărcare a secţiunii în care sunt definite.

Segmentarea programului este procedeul de împărţire a unui program în segmente, astfel încât să fie posibil ca în timpul execuţiei programului să fie încărcat permanent, în memoria internă, numai segmentul rădăcină, segmentele subordonate putând să fie încărcate pe rând, prin reacoperirea segmentelor între care nu există raporturi de subordonare, sau să fie executate în paralel.

Pentru aceasta, odată cu generarea programului executabil, editorul de legături construieşte şi tabela de legături asociată, în care sunt memorate informaţii despre :

o numele segmentului rădăcină şi a celorlalte segmente

o adresele segmentelor, în programul executabil, relativ la adresa 0 de memorare a programului executabil

o lungimea fiecărui segment

o punctul de intrare în program, adică adresa primei instrucţiuni executabile din program.

Editorul de legaturi produce o versiune intermediara a programului, care este normal scris intr-un fisier sau o librarie pentru o executie mai tarzie. Editorul de legaturi realizeaza realocarea tuturor sectiunilor de control relationate de la inceputul programului relationat.

Activitatile aditionale ale editorului de legaturi:
- inlocuieste subrutine in programul relationat;
  de ex:
    INCLUDE PLANNER(PROGLIB)
    DELETE PROJECT {DELETE from existing PLANNER}
    INCLUDE PROJECT(NEWLIB) {INCLUDE new version}
    REPLACE PLANNER(PROGLIB)
-constuieste un pachet de subrutine utilizate in general impreuna;
- nu poate evita memorarile multiple ale librariilor obisnuite in programe;
- are nevoie de un incarcator de legaturi pentru a combina librariile obisnuite la timpul executiei;

Parte tratata de Ionut Ciotir(441A):

LINKER

PROCESUL DE *LINKING*

In termeni tehnici un liker sau un editor de legatura este un program care imbina obiectele* generate de compilator si librariile** si le imbina intr-un fisier executabil sau intr-un fisier dynamik-link library (DLL). Pe sisteme UNIX termenul *loader* este folosit ca sinonim pentru *linker*. Totusi termenul de *loader* este ambiguu , definind o arie mai restransa de procese, de multe ori insemnand doar incarcarea in memroie ai unui program de pe hard in memoria principala si pregatirea pentru rulare.

Procesul de *linking* este format din doi pasi.

In primul pas linker-ul ia ca date de intrare librarii (importante sau standard), resurse , definitii de module. Linker-ul multor limbaje nu foloseste extensia fisierelor pentru a determina continutul, ci examina fiecare fisier de intrare pentru a determina ce fel de fisier este si cum v-a fi folosit. Fiecare data de intrare (fisier) va contine minim un marker, in general reprezentand rutinele ce vor fi apelate, si scopul acestor. Aceste markere sunt de doua tipuri:

- markere definite sau exportate , acestea sunt functii sau variabile ce sunt presente in secventele prezente intr-un obiect , si pot fi apelate de alte secvente (public)
- markere nedefinite sau importate, fiind functii sau variabile ce sunt apelate prin referinte de catre obiect dar nu sunt definite intern (in secventa de cod)

Dupa ce a determinat rolurile fisierelor editorul de legaturi creaza un tabel ce va contine tot ce a fost definit in procesul de intrare, si un tabel ce contine markerele intrarilor.

In cel de-al doilea pas se folosesc informatile acumulate in timpul primului pas pentru a se

face efectiv operatia de legatura. Transforma codul , transformand adresele definite in referinte catre obiecte si aloca memoria definita in cod. Apoi este creat fisierul de iesire, care contine tabelele de referinta. De multe ori *linkerul* genereaza portiuni mici de cod in fisierul de iesire , numit "*glue code"* , cod folosit la apelul rutinelor in momentul apelarii programului.

Unele fisiere create pot fi din nou folosite pentru o noua legatura, daca fisierele contin proprietatile unui fisier de intrare (marker).

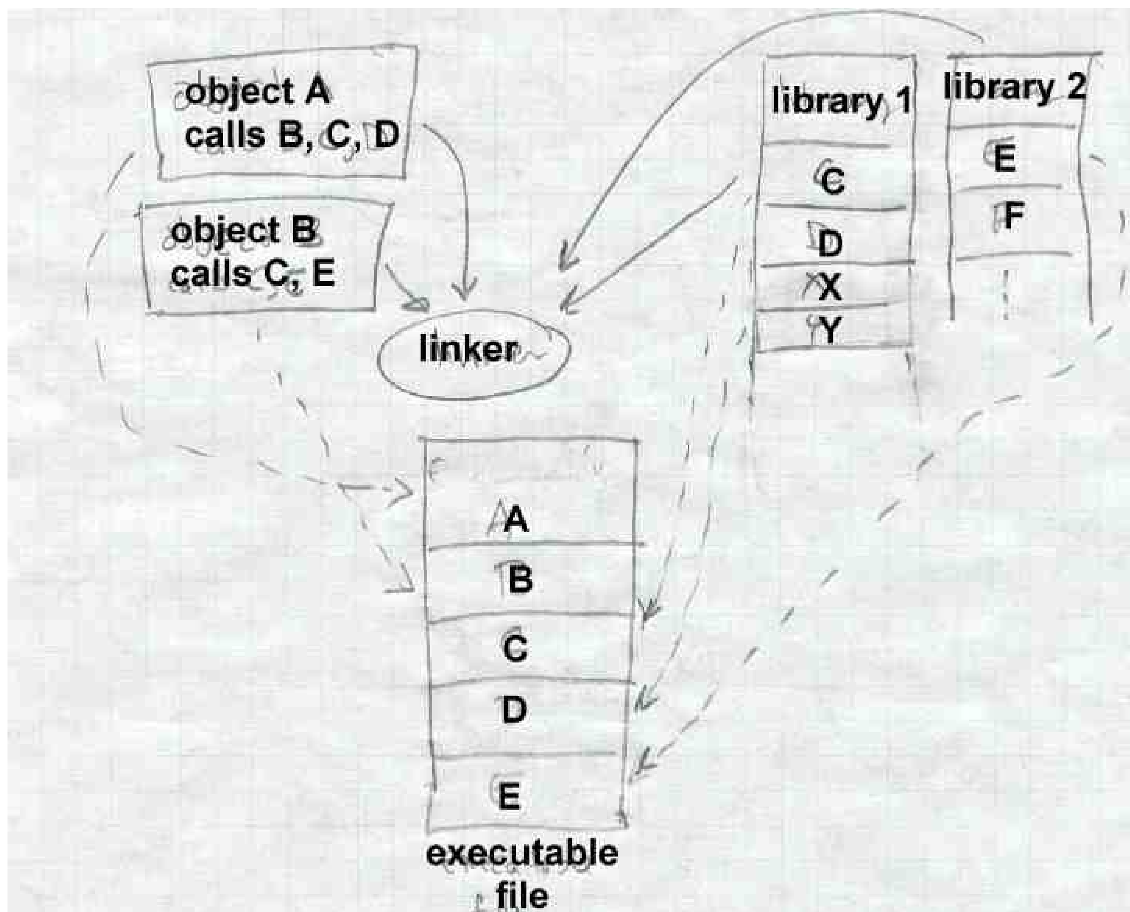
Toate formatele au posibiliatatea folosirii unor identificatori pentru *debugging*(control al erorii) pentru ca atunci cand programul este rulat sub controlul unui *debugger* programatorul poate controla numerele si denumirile folosite in cod-ul sursa.

Exista *linkere* care lucreaza intr-un singur pas, folosindu-se de meorie sau de disc pentru a incarca continutul datelor de intrare folosite in timpul procesului de *linkare*, proces ce se numeste *buffering ,* datele fiind accesate din *buffer* mai tarziu.

----

* Obiectele sunt secvente de program care contin cod masina si informatii pentru *liker*.

** O librarie este o colectie de subprograme(functii) folosite pentru diverse aplicatii. Librariile contin "helper"-e si cod-uri care contine servicii pentru programe independente (ex: functii apelabile in C++ la includerea anumitor librarii)



LIBRARII STATICE

O librarie statica , sau arhiva , consta intr-o serie de rutine care sunt copiate intr-o aplicatie de compilator , elementul de legatura (*linker)* sau legatorul (*binder)* , producand fisiere obiect si executabile de sine statatoare. Procesul , impreuna cu executabilul rezultat se numeste constructie statica (*static build)* a aplicatiei. *Likerul* rezolva toate adresele necunoscute in adrese realocate prin incarcarea completa a codului si a librariilor in momentul executiei locatiile de memorie.

Procesul de *linking* poate lua mai mult decat procesul de compilare , si trebuie refacut de fiecare date cand un modul este recompilat. Un *linker* poate merge numai pe anumite tipuri de obiecte , si deci cerand librarii specifice (si compatibile). Procesul de legare (*linking process)* se foloseste de referinte in ordinea data. Nu este considerata o eroare un nume ce poate fi gasit de mai multe ori intr-un set de librarii.

LEGATURI DINAMICE

Sistemele de operare din ziua de azi permit legarea dinamica. Legarea dinamicã inseamna cã subprogramele unei biblioteci sunt incarcate intrun program - aplicatie la momentul executiei, decat sa fie legate la compilare , ramamand ca fisiere separate pe disc. Acesta inseamna ca executabilele inca mai contin simboluri neidentificate, plus o lista de obiecte sau librarii care ofera definitii pentru acestea. La rularea programului se vor incarca obiectele si librariile.

Aceasta abordare ofera doua avantaje:

- Librariile des folosite (de exemplu librariile standard ale sistemului) trebuiesc stocate intr-o singura locatie;
- Daca o eroare intr-o functie a unei librarii este corectata prin inlocuirea librariei, toate programele care o folosesc dinamic vor beneficia de corectie dupa restartarea lor. Programele care inculd aceste functii prin legare statica vor avea nevoie de refacerea legaturilor.

Parte tratata de Adrian Voinea((441A)

„RELAXAREA” EDITORULUI DE LEGATURI

Intrucat compilatorul nu are nici o informatie asupra ordinii obiectelor in ceea ce priveste rezultatul final, nu poate profita de instructiuni mai scurte si mai eficiente. De exemplu, o instructiune de salt se poate referi fie la o adresa exacta, fie la o variatie de la locatia curenta, iar decalarea poate fi exprimata prin diferite marimi ce depind de distanta pana la locatia respectiva. Acesta etapa poate fi realizata doar dupa ce toate obiectele introduse la inceput au fost citite si carora l-i s-au atribuit adrese temporare; ulterior relaxarea reatribuie adrese care in schimb pot permite mai multe relaxari ulterior.

In general secventele substituite sunt mai scurte, ceea ce permite acestui proces ca intotdeauna sa urmeze solutia optima avand in vedere o ordine corecta a obiectelor. Daca acest lucru nu se intampla, relaxarea nu se poate produce, iar editorul de legaturi trebuie sa aleaga o alta optiune.

PROGRAME DE COMANDA

Fiecare *linker* are un program de comanda pentru a controla procesul de legare. La finalul procesului , *liker*-ul are nevoie de lista obiectelor si librariilor de legatura. In general exista o lista de optiuni: fie sa se pastreze markerele de *debugging* sau sa se foloseasca biblioteci comune sau necomune, care din posibilele rezultate sa fie folosite. Marea majoritate a programelor de legare permit o metoda pentru a specifica adresa unde codul de legare va fi pus, ce vine la indemana cand este folosit un *liker* pentru a lega un fisier *system kernelsystem kernel* sau alte programe care nu necesita controlul unui sistem de operare. Linkere-le care suporta coduri multiple pot folosi o comanda care specifica in ce ordine fisierele sa fie legate, putandu-se specifica diferite optiuni.

Exista patru tehnici pentru a “comanda” un *linker*:

- *Command line:* linkerele au o line de comanda sau un echivalent., prin care se pot da comenzi pentru nume de fisiere , si schimbari. Aceasta metoda este folosista de *linkere*-le de pe sistemele Unix si Windows. Pe sistemele cu lungime limitata de comenzi exista o cale de a face linker-ul sa citeasca comenzile dintr-un fisier comportandu-se ca si cum ar fi comenzi date din linia de comanda.
- *Intermixed with object files:* Unele linkere , cum ar fi likerele de mainframe IBM, accepta obiecte si comenzi alternative intr-un singur fisier de intrare. (acest procedeu dateaza de pe vremea calculatoarelor cu cartela)
- *Embedded in object files:* Unele formate de obiecte (cele Microsoft) permit comenzi de legare sa fie scrise in interiorul fisierelor obiect. Acest lucru permite compilatorului sa transmita orice optiune necesara legarii unui obiect in fisierul in sine. (ca exemplu compilatorul de C trimite comezi pentru cautarea in librariile standard ale C-ului)
- *Separate configuration language:* Mai rar unele *linkere* au integrate un program intreg configurat pentru control, care lasa programatorul sa specifice in ce ordine sa fie incarcate (legate) segmentele , regului pentru combinarea segmentelor asemanatoare, adresele segmentelor , si o gama larga de alte optiuni.

LIBRARII COMUNE SI PROGRAME

In sistemele care lucreaza cu multiple programe simultan , fiecare program are un set propriu de tabele, programele avand adrese logice separate. Suport considerabil din partea linkerului este necesar pentru ca sa mearga sharing-ul. In programul executabil *linker*-ul trebuie sa grupeze tot cod-ul executabil intr-o singura parte a fisierului, care fa deveni read-only. Cand diferite programe folosesc o librarie comuna linker-ul trebuie sa marcheze fiecare program , ca atunci cand fiecare incepe libraria este facuta o referinta in adresa programului.

Bibliografie:

- http://www.iecc.com/linker/
- microsoft MSDN
- en.wikipedia.org

Parte tratata de Alexandru Persu(443A)

c) Structura unui modul(fisier) obiect

Compilatoarele si assembler-ele creaza fisiere obiect care contin codul binar generat si informatia pentru un fisier sursa. Linkerele combina multiple fisiere obiect intr-unul singur , loader-ele iau fisierele obiect si le incarca in memorie.(Intr-un mediu de programare integrata, compilatoarele, assemblerele, si linkerele ruleaza implicit cand utilizatorul le spune sa construiasca un program).

Ce contine un fisier obiect?

Un fisier obiect contine cinci tipuri de informatie.

- **Informatia Header:** informatia completa despre fisier: ca lungimea codului, numele fisierului sursa, si data cand a fost creat.
- **Codul obiect:** instructiuni binare si informatiile generate de un compilator sau de un assembler.
- **Relocarea:** o lista de locuri in codul obiect, care trebuie sa fie fixate bine cand linker-ul schimba adresele codului obiect.
- **Simboluri:** simbolurile globale definite in acest modul, simbolurile ce vor fi importate din alte module sau definite de linker.
- **Informatii de depanare:** alte informatii despre codul obiect care nu sunt necesare pentru linking, dar necesare pentru un debugger. Asta include informatia despre fisierele sursa si despre numarul liniei, simbolurile locale, descrierile structurilor de date folosite de codul obiect ca definitiile structurilor C.

Observatie

Nu toate formatele obiect contin aceste tipuri de informatii, si este posibil sa avem formate destul de folositoare cu informatie putina sau chiar deloc, dincolo de codul obiect.

Structura unui modul(fisier) obiect

Formatul unui fisier obiect este formatul unui fisier computer folosit pentru stocarea de cod obiect si date specifice produse de un compiler sau un assembler.

Proiectarea unui format obiect este un compromis condus de mai multe utilizari la care un fisier obiect este supus. Un fisier poate sa fie *"linkable"*, folosit ca intrare de un editor de link sau de un linking loader. Poate fi *executabil*, capabil sa fie incarcat in memorie si sa ruleze ca un program, *loadable*, capabil sa fie incarcat in memorie ca o librarie alaturi de un program, sau orice combinatie a celor trei. Unele formate suporta doar una sau doua dintre aceste utilizari, altele suportandu-le pe toate trei.

Un fisier linkable contine simbol extensi si informatie de relocare , necesara linker-ului impreuna cu codul obiect. Codul obiect este de obicei impartit in multe segmente mici, logice, care vor fi tratate diferit de catre linker. Un fisier executabil contine cod obiect, de obicei pagina aliniata pentru a permite fisierului sa fie mapat intr-un spatiu de adresa, dar nu are nevoie de nici un simbol(decat daca se face rularea in linking dinamic) si are nevoie de putina sau nici o informatie de relocare. Codul obiect este un segment unic mare sau un set mic de segmente care reflecta mediul de executie al hardware-ului, cel mai intalnit read-only vs. read-write a paginilor. Depinzind de detaliile unui mediu de rulare a sistemului, un fisier incarcabil, poate consta exclusiv dintr-un cod obiect sau poate contine un simbol complet si informatie de relocare pentru a permite linking-ul timpului de executie simbolic.

Intre aceste aplicatii exista cateva conflicte.Orientarea logica a grupurilor orientate a segmentelor 'linkable" rare ori se potrivesc cu grupurile orientate hardware ale segmentelor executabile. In particular pe computer'erele mai mici, fisierele "linkable" sunt citite si scrise pe rand de catre linker, in timp ce fisierele executabile sunt incarcate in intregime in memoria principala. Aceasta deosebire este mai evidenta in formatul complet diferit, MS-DOS "linkable" OMF si formatul executabil EXE.

Sunt multe formate de fisier obiect diferite; la inceput fiecare computer avea unicul sau format, dar odata cu avansarea sistemului UNIX si a altor sisteme de operare portabile , unele formate gen COFF si ELF, au fost definite si folosite pentru alte tipuri de sisteme. Este obisnuit ca acelasi tip de format de fisier sa fie folosit atat ca intrare si iesire pentru liker, cat si ca librarii si **fisiere cu format executabil**.

Design'ul si/sau alegerea unui format de fisier obiect este o parte cheie asupra design'ului de sisteme. Acesta alegere afecteaza performantele linker'ului si astefel dispozitivul de programare este nevoit sa se intorca din rulare. Daca formatul este folosit pentru executabile, design'ul de asemenea afecteaza timpul necesar unui program pentru a incepe rularea, si astfel raspunsul catre utilizator. Majoritatea formatelor de fisier obiect sunt structurate ca blocuri de date, fiecare bloc continand un anumit tip de data. Aceste blocuri pot fi paginate datorita nevoii sistemului de memorie virtuala, astfel ne mai fiind nevoie de alte procese pentru a putea fi folosit.

Cel mai simplu format de fisier obiect este formatul DOS.COM, care este un simplu fisier (ce contine un sir de bytes neprelucrati) ce este intotdeauna incarcat la o adresa fixa. Alte formate contin o inlantuire elaborata de structuri si substructuri de caractere, ale caror specificati ruleaza pe mai multe pagini.

Tipuri de date suportate de formatele de fisier obiect tipice:
- BSS (Block Started by Symbol);
- segmente text;
- segmente de date;

Formate notabile de obiect

- DOS
  - .COM
  - DOS executableDOS executable (MZ)
  - Relocatable Object Module FormatRelocatable Object Module Format (cunoscute ca "OBJ file" or "OMF"; de asemnea folosite in Microsoft Windows de catre unele unelte)
- Embedded
  - IEEE-695
  - S-records
- Macintosh
  - PEFPEF/CFMPEF/
  - Mach-OMach-O (NEXTSTEPMach-O (Mach-O (, Mac OS XMach-O (, Mach-O (,)
- Unix
  - a.out
  - COFFCOFF (System VCOFF (COFF ()
    - ECOFFECOFF (MipsECOFF (ECOFF ()
    - XCOFFXCOFF (AIXXCOFF (XCOFF ()
  - ELFELF (SVR4ELF (ELF (; folosite in majoritatea sistemelor folosite)
  - Mach-OMach-O (NeXTMach-O (Mach-O (, Mac OS XMach-O (, Mach-O (,)
- Microsoft Windows
  - 16-bit New Executable
  - Portable ExecutablePortable Executable (PE)
- Others
  - IBM 360 object format

- NLM
- OMFOMF (VME)
- SOMSOM (HP)
- XBEXBE - XboxXBE - XBE - executabil
- APP - Symbian OS format de fisier executabil
- RDOFF
- HunkHunk - AmigaOSHunk -

Bibliografie:

- en.wikipedia.org
- http://www.iecc.com/linker/

Subpunct tratat de Vlad Noghi(443A)

Legatura dinamica(comparatie linux vs. windows)

Aceasta tema dezbate conceptul librariilor share-uite in cele 2 sisteme de operare cel mai des intalnite Windows si Linux, si ofera o incursiune printre structurile de date variate pentru a explica cum legarea dinamica este facuta in aceste sisteme de operare. Lucrarea va fi folositoare pentru programatorii interesati in implicatiile legate de securitate si viteza relativa a legaturii dinamice, si presupune niste cunostinte anterioara a acestui subiect.

In cele ce urmeaza vom discuta despre cum functioneaza in wondows iar mai apoi vom continua sa comparam cele 2 medii.

Structurile de date cu fisiere executabile portabile Windows (PE)

Stim ca o sectiune este o bucata de cod sau informatie legata intr-un mod logic, si mai stim ca informatia unor tabele de importuri a unui executabil sunt intr-o sectiune. In acest referat ne uitam la niste sectiuni in fisierele Windows PE.

Sectiunea de exporturi(.edata)

Sectiunea .edata incepe cu structura de sirectoare export IMAGE_EXPORT_DIRECTORY.

Directorul de exporturi contine RVA-urile(adresele virtuale relative) ale

Tabelei de Adrese de Export. Acesta contine adrese ale punctelor de intrare exportate, informatie exportata si valori absolute. Un numar ordinal este folosit pentru indexarea tabelei de adrese. BAZA ORDINALA trebuie sa fie extrasa din numarul ordinal inainte de indexarea inauntrul tabelei.

Pointeri tabele de nume export: Acest vector contine adrese in tabele de nume export. Pointerii sunt legati la baza imaginii si sunt ordonati lexical pentru a inlesni cautarea binara. Tabela de nume export contine nume(ASCII) pentru intrarie exportate in imagine.

Tabela ordinala de export: Pointerii tabelei de nume de export si tabela de export ordinala din cei 2 vectori paraleli. Vectorul tabela ordinala de export contine numarul ordinal asociat cu numele exportat referit de pointerii tabela de nume de export. Numarul ordinal va servi ca index in EAT.

Sectiunea de importuri(.idata)

Sectiunea .idata face conversia a ceea ce face sectiunea .edata, descrisa mai sus. Ea mapeaza simboluri/numere ordinale inapoi in RVA-uri. Sectiunea .idata incepe cu o tabela de directoare de importuri IMAGE_IMPORT_DIRECTORY . Tabela de directoare de importuri este compusa dintr-un vector de structuri IMAGE_IMPORT_DESCRIPTOR, cate unul pentru fiecare executabil importat. IMAGE_IMPORT_DESCRIPTOR contine RVA-uri de tipul:

Tabela de referinte de import: Aceasta este un vector de structuri IMAGE_THUNK_DATA. Structura contine numerul ordinal sau numele RVA pentru fiecare functie importata. Tabela identifica simbolurile de importat, cu intrarile in tabela import de referinta fiind paralel cu acelea din tabela de adrese import(IAT). Daca bitul cel mai semnificativ este setat atunci ceilalti biti reprezinta sunt numarul ordinal. Altfel intrarea este RVA-ul unei intrari in tabela de referinta pentru nume .

Tabela de adrese de import: Aceasta este tot un vectori de structuri IMAGE_THUNK_DATA. Initial amandoua tabela de referinta(Lookup table) si tabela de adrese import(IAT) contin intrari simileare. Loader-ul introduce adresele fiecarei rutine importate in aceasta tabela, in timp ce intrarile din tabela de referinta a intrarilor(Import Lookup Table) retine informatia originala cum era initial. Vom vedea de ce linker-ul mentine informatia originala mai tarziu cand vom discuta despre operatia numita binding.

Tabela de nume indiciu(Hint-Name Table): Tabela contine un indiciu de 4 byte-uri urmat de un simbol nume nul. Valuarea indiciului este folosit pentru indexarea vectorilor de pointeri catre Tabela de nume de export(Export Name Table) permitand importuri mai rapide. Indiciul va fi corect daca DLL-ul nu s-a schimbat sau cel putin lista sa de simboluri exportate nu s-a schimbat. Daca indiciul este incorect atunci cautarea binara is facuta in Tabela de pointeri nume(Export Name Pointer table).

Cum functioneaza lucrurile

Incarcand un executabil de Windows si un DLL este simimlar cu incarcarea unui program ELF legat dinamic in Liunx. Diferenta este ca link-erul este o parte a kernel-ului insusi. Prima data kernel-ul mapeaza in executabilul indrumat de header-ele PE. Loader-ul se uita la IAT al modulului si determina daca DLL-ul depinde de DLL-uri aditionale, si daca loader-ul le mapeaza si pe acestea.

Acest proces continua pana cand toate modulele dependente au fost mapate in meorie.

O functie importata poate fi listata dupa nume sau dupa numarul ordinal. Numarul ordinal reprezinta pozitia ei in Tabela de adrese exportate al DLL-ului. Daca este listata de nume, loader-ul face o cautare binara a Tabelei de pointeri nume exportate a DLL-ului corespunzator pentru a cauta index-ul la care simbolul este gasit.Dupa aceea el foloseste acel index ca un index in Tabela de numere ordinale exportate pentru a primi numarul ordinal care este folosit ca un index in Tabela de adrese exportate. Adaugand RVA-ul simbolului gasit din Tabela de Adrese de export (EAT) la adresa de incarcare a DLL-ului va rezulta adresa absoluta pe care loader-ul o scrie la intrarea corespunzatoare in tabela de adrese de import (IAT)

Incarcarea lenesa in Windows

Un DLL incarcat cu delay are o structura ImgDelayDescr similara cu structura de directoare de date de import .idata dar el nu este in sectiunea .idata. ImgDelayDescr contine adresele unei tabele de adrese de import(IAT) si a unei tabele de nume de import(INT) pentru DLL. Aceste tabele sunt identice in format cu cele normale de importuri, dar mai degraba ele sunt scrise si citite de codul librariilor runtime decat sistemul de operare. Cand apelezi un API pentru prima oara dintr-un DLL incarcat cu delay, libraria runtime incarca DLL(daca este necesar), primeste adresa, si depoziteaza tabela de adrese de import incarcata cu delay astefl incat apelarile viitoare merg direct la fisierul API.

Zona tratata de Horia Noghi(443A):

Librarii statice sau Librarii comune

O librarie este o colectie de sub-programe care permite codurilor sa fie publice si astfel sa fie schimbate intr-o anumita maniera. Executabilele si librariile fac referinte catre altele prin link-uri.

In cel mai comun mod posibil, librariile pot fi impartite in doua categorii: statice si comune.

Librariile statice sunt o colectie de fisiere obiecte si in mod conventional eu extensia „.a” in variantele UNIX si „.lib” in Windows. Cand un program are o legatura (link) impotriva unei librarii statice, codul masina din fisierele obiecte pentru orice functii externe folosite de program este copiat din librarie in executabilul final.

In contrast cu librariile statice, cu librariile comune, codul librariei nu este conectat la executabil in moemntul efectuarii legaturii. Depinzand de cand sau cum legaturile adreselor sunt facute, procesul de atasare poate fi categorisit in : prelinking, momentul efectuarii legaturii, legatura implicita si explicita.

Codul independent ca pozitie (Win32 DLLs sau ".SO")

Acest cod poate fi copiat in orice locatie de memorie fara modificari si dupa aceea executat spre deosebire de codul dependent de pozitie care necesita procesari speciale de un utilizator pentru a-l face executabil intr-o alta

locatie.

Win 32 DLL-urile nu sunt ca pozitie independente. Ele au nevoie de astfel de procese in timpul incarcarii. Mutarea in aceeasi adresa poate fi comuna, dar daca procese diferite au conflicte cu layout-urile memoriei, utilizatorul trebuie sa genereze copii multiple ale acestor DLL-uri in memorie. Cand un incarcator Windows copiaza un DLL in memorie, el deschide fisierul si incearca sa copieze fisierul in memorie si adresa sa de baza. Cand paginile copiate sunt accesate sistemul de paginare va vedea daca paginile sunt deja prezente in memorie. Daca ele sunt, ele doar aloca aeste pagini noului proces deoarece copierea a fost deja executata de incarcator la adresa de baza preferata. Altfel fisierele sunt incarcate de pe disc.

Daca adresa preferata a DLL-ului nu este disponibila, incarcatorul copiaza fisierele intr-o locatie libera a spatiului adresei procesului. In acest caz, el marcheaza codul paginii astfel: COW(copy-on-write) care inainte era marcat ca si read-execute, devreme ce incarcatorul va trebui sa ruleze codul la momentul relocarii.

Linux rezolva aceasta problema folosind PIC-urile(Position Independent Code). Obiectele comune in Linux de obicei contin PIC-uri care evita necesitatea de a copia libraria in momentul incarcarii. Toate fisierele de cod pot fi share-uite impotriva tuturor proceselor folosind aceeasi librarie si pot fi incarcate/descarcate din sistemul de fisiere. In x86, nu exista nici o cale simpla de a adresa date relative in locatia curenta devreme ce toate apelarile sunt relationate cu instructiuni de tip pointeri. De aici incolo, toate referintele catre globalele statice erau directionate printr-un „table” numit Global Offset Table(GOT).

Legaturile dinamice in Linux

Structurile de date ELF

Vom discuta in mare doar despre acele structuri care sunt relevante pentru acest proiect. Pentru legaturile dinamice, ELF foloseste 2 tipuri de astfel de table-uri specifice procesoarelor, si anume GOT si Procedure Limkage Table(PLT).

Global offset Table (GOT)

ELF suporta coduri PIC prin intermediul GOT-ului in fiecare librarie comuna. GOT-ul contine adresele absolute ale tuturor datelor statice relevante in program. Adresa GOT-ului este in mod normal inclusa intr-un registru (EBX) care este o adresa relativa din cod care ii fac referire.

Procedure Linkage Table (PLT)

Si executabilul care foloseste librariile comune si librariile comune insesi au un PLT. Astfel, pecum GOT-ul redirectioneaza orice calcul de adresa independenta de pozitie catre locatii absolute, PLT-ul redirectioneaza apelari de functii de pozitie independente catre locatii absolute.

Inafara de aceste doua tipuri, limk-erul se refera si la .dynsym, care contine toate simbolurile fisierelor importate si exportate , .dynstr, care contine numele string-urilor simbolurilor, .hash care contine tabla hash la care limk-erul are acces pentru a cauta simboluri rapid si .dynamic, care reprezinta o lista de valori insemnate si pointeri.

In sectiunea .dynamic insemnurile importante sunt:

DT_NEEDED: acest element tine tabla de stringuri offset al unui string null-terminat string, dandu-i numele unei librarii necesare. Offset-ul este un index in tabla inregistrata in DT_STRTAB.
DT_HASH: aceasta tine adresa simbolului „hash table" si se refera la simbolul „table" la care se face referire din partea elementului DT_SYMTAB.
DT_STRTAB: Tine adresa tablei sirurilor de caractere.
DT_SYMTAB: tine adresa tablei simbolurilor.