Static Linking در برابر Dynamic Linking

محمد شمس جاوی
http://www.mshams.ir

چکیده

کاربرد لینکرها و ساختار آنها یکی از مباحث کلاسیکی است که پیدایش آن به دوره پیشرفت کامپیوترها و پیدایش سیستم عاملها باز می‌گردد. در این مقاله پس از معرفی چگونگی ظهور کتابخانه‌های اشتراکی، به معرفی روشهای لینک ایستا و لینک پویا در لینکرها پرداخته شده و پس از معرفی مزایا و معایب هرکدام، با یکدیگر مقایسه میگردند.

همچنین ساختار فایلهای کتابخانه‌ای DLL به عنوان نماینده روش لینک پویا، معرفی شده و طریقه اعلان و استفاده از آنها به روش ایستا و پویا، تشریح می‌گردد..

مقدمه:

در ابتدای پیدایش کامپیوترها، برنامه نویسان برنامه‌های خود را مکتوب کرده و سپس معادل کد ماشین آنها را بدست آورده و در کامپیوترهای اولیه ثبت می‌کردند. روند ثبت، گاهی پانچ کردن کارتهای سوراخدار جهت اجرای دسته‌ای کارها (Batch) روی کامپیوترهای مرکزی و گاهی نیز ثبت و ذخیره مستقیم آنها توسط IC Programmer ها روی حافظه ریزپردازنده یا کامپیوتر بود.

در صورتی که برنامه نویسان از آدرسهای نمادین جهت پرش به نقاط مختلف، در برنامه‌های خود استفاده می‌کردند، خود وظیفه ترجمه و تبدیل آن آدرسها به مقادیر حقیقی (آدرسهای فیزیکی) را در حین تبدیل به کد ماشین بر عهده داشتند.

پس از آن با پیدایش و پیشرفت محیطها و زبانهای برنامه نویسی، اسمبلرها این وظیفه را به عهده گرفته و برنامه نویسان می‌توانستند برنامه‌های خود را در قطعات مختلف و مجزا نوشته و از آدرسدهی نمادین یا نسبی استفاده نمایند، چرا که اسمبلرها آدرسهای مذکور را به آدرسهای فیزیکی تبدیل کرده و پس از هر اعمال تغییراتی در برنامه، تنها کافی بود که یکبار دیگر آن برنامه را اسمبل کرد.

کتابخانه‌های اشتراکی (Shared library):

پیدایش کتابخانه‌های اشتراکی به زمانی قبل از استفاده از اسمبلرها مربوط می‌شود. اولین بار در سال 1947، John Mauchly مبدع پروژه ماشین رمزنگاری ENIAC استفاده از برنامه‌های از پیش نوشته شده و ذخیره شده روی نوارهای مغناطیسی در یک برنامه جدید را مطرح کرده و به تشریح دلایل برتری این کار، و نیاز به ترجمه و تبدیل آدرسهای بارگذاری قطعه برنامه‌های مذکور پرداخت.

وی نیاز به ساخت دو روال استاندارد، آدرسدهی مجدد جهت جابجایی کدها (Relocation) و جستجوی کتابخانه‌ها (Library Search) را مطرح کرده و پس از ساخت آنها کلیه قطعه برنامه‌های خود را به زبان ماشین و با شروع از آدرس صفر به عنوان آدرس بارگذاری آن زیر برنامه (Sub program) نوشت. بدین شکل در هنگام نرجمه برنامه اصلی، تمام قطعه برنامه‌های مورد استفاده جستجو شده و توسط روال Relocation، آدرس شروع آنها نسبت به مکان قرارگیریشان در کد ماشین برنامه، اصلاح می‌شد.

به این ترتیب مفهومی به نام کتابخانه اشتراکی (Shared library) شکل گرفت که بیان می‌کرد تمام برنامه‌هایی که از قطعه کدهای مشابه موجود در محتویات یک کتابخانه استفاده می‌کنند، می‌توانند یک نسخه واحد از آن را مشترکا مورد استفاده قرار بدهند.

لینکرها (linkers):

با ظهور سیستم عاملها، تحولات بسیاری در زمینه ساختار برنامه‌ها و نحوه اجرا و بارگذاری آنها پدید آمد. در گذشته هر برنامه تمام حافظه ماشین و پردازنده را به تنهایی در اختیار داشته و همه برنامه‌ها، با توجه به آدرسهای ثابت حافظه که در اختیارشان قرار گرفته بود ترجمه می‌شدند. اما با پیدایش سیستم عامل این امر امکان پذیر نبود، چرا که حافظه مورد استفاده مشترک میان برنامه، سیستم عامل و دیگر برنامه‌ها قرار می‌گرفت و بدین شکل تا زمانی که برنامه مذکور توسط سیستم عامل در حافظه بارگذاری نمی‌شد، آدرسهای حقیقی مربوط به بارگذاری آن مشخص نبودند.

بدین شکل وجود مفاهیمی مانند linker ها و loader ها مورد اهمیت بیشتری قرار گرفتند.

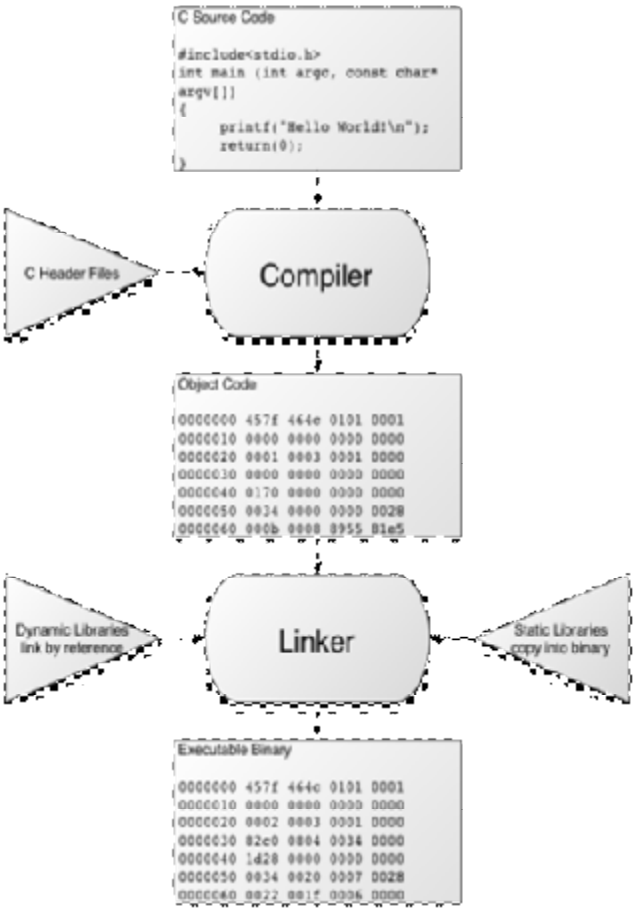
عملیات لینک (Linking):

در ساده‌ترین حالت، لینکر با فایلهای Object تولید شده در آخرین مرحله کامپایل سروکار داشته و آنها را به یکدیگر متصل کرده و در نهایت یک فایل واحد تولید می‌گرداند. فایلهای Object حاوی کدهای ماشینی هستند که کامپایلرها یا اسمبلرها پس از پردازش کدهای منبع (Source Code) تولید می‌کنند.

به عنوان مثال، یکی از انواع رایج و شناخته شده فایلهای Object، فایلهای اجرایی COM در سیستم عامل MS-DOS هستند که حاوی کدهای خام ماشین بوده و فاقد هر گونه ساختار و قالب‌بندی از پیش تعیین شده می‌باشند.

پروسه لینک در واقع آخرین مرحله از روند کامپایل یک برنامه محسوب می‌شود. لینکر کدها و قطعه برنامه‌های مورد ارجاع توسط یک برنامه را به شکل ماژولهایی مناسب به برنامه اضافه کرده و یک فایل مناسب جهت بارگذاری و اجرا در حافظه تهیه می‌نماید، چرا که هر برنامه جهت اجرا می‌بایست از حافظه جانبی به حافظه اصلی کپی شده و آدرسهای مورد استفاده و ارجاع در آن به آدرسهای حقیقی ترجمه شوند. حال اگر این برنامه از مجموعه‌ای از زیربرنامه‌های جانبی استفاده کرده باشد، تمامی آنها نیز می‌بایست به شکل مناسبی به برنامه اصلی اضافه شده یا به هر شکل در هنگام اجرای برنامه مورد دسترسی و ارجاع قرار بگیرند.

به عنوان مثال اگر برنامه ما در زبان پاسکال از تابع sqrt() موجود در کتابخانه Math استفاده کرده باشد، برای دسترسی به کد بدنه تابع، می‌بایست ارجاع آن فراخوانی به کتابخانه مذکور را بررسی کرده و مکان کد تابع را در کتابخانه پیدا کنیم. پس از این کار می‌بایست دستور فراخوانی تابع به نوعی Patch (وصله) شده یا اینکه کد تابع به برنامه اصلی ضمیمه گردد تا در زمان اجرا بتوان از آن استفاده نمود.



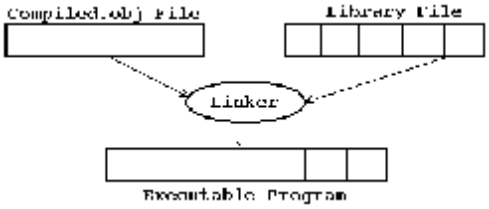
روند کار Linker و Compiler

پس در حقیقت کار اصلی لینکرها، ویرایش لینکها و ارجاعات موجود در برنامه به شکلی مناسب جهت اجرا می‌باشد. عمل لینک به دوشکل ایستا و پویا مورد استفاده قرار گرفته که در ادامه تشریح می‌شوند.

لینک ایستا (Static Linking):

در این روش تمامی موارد ارجاع برنامه، از کتابخانه مربوطه پیدا شده و در برنامه اصلی کپی می‌شوند. به عبارتی یک کپی از کتابخانه‌های مورد نیاز تهیه شده و به طور فیزیکی به برنامه الحاق می‌گردد. برخی از مزایای این روش به شرح زیرند:

- برنامه حاصل، یک برنامه واحد بوده و قابلیت حمل بالایی دارد. یعنی که می‌توان بدون نگرانی برنامه فوق را روی دیگر سیستمها نیز اجرا کرد، زیرا که تمامی کدها و اطلاعات مورد نیاز برنامه به آن ضمیمه شده‌اند.
- سرعت اجرای این برنامه بیشتر از حالت پویاست. البته اختلاف سرعت در برنامه‌های عادی محسوس نبوده، اما به دلیل در دسترس بودن تمام قسمتهای مورد نیاز در کد برنامه، دیگر نیاز به ارجاعات و بارگذاریهای زمان اجرا ندارد، فلذا با سرعت بیشتری اجرا می‌شود.



لینک ایستا

البته معایب این روش هم کاملا آشکار بوده و به شرح زیرند:

- به دلیل اضافه شدن روتینهای مورد ذکر به برنامه، حجم فایل نهایی (Executable image) افزایش یافته و بارگذاری آن در حافظه هزینه بیشتری را به سیستم تحمیل می‌کند.
- در صورت اجرای چندین نسخه از برنامه، تمامی آنها حاوی تمام روتینهای ضمیمه شده مورد ارجاع بوده و این سربار حافظه، به همه آنها تحمیل گشته و مصرف حافظه بسیار زیاد می‌گردد.
- در صورت اعمال کوچکترین تغییری در کتابخانه‌های مورد استفاده، فایل اجرایی تمام برنامه‌هایی که از محتوای آن کتابخانه‌ها استفاده کرده‌اند می‌بایست دوباره اسمبل و کامپایل گردد.

کتابخانه‌های ایستا به نام آرشیو (Archive) هم شناخته شده و در برخی زبانها با پسوند ".a" مورد استفاده قرار می‌گیرند. توابع و زیربرنامه‌های موجود در آنها پس از الحاق به برنامه به همراه بدنه خود

برنامه در فایل Object قرار گرفته و سپس این فایل به کد ماشین ترجمه می‌گردد.

لینک پویا (Dynamic Linking):

پس از پیشرفت کامپیوترها و زبانهای برنامه‌نویسی، افزایش روز افزون حجم برنامهای تولیدی موجب بوجود آمدن مشکلات جدی در زمینه بارگذاری برنامه‌ها در حافظه‌های محدود کامپیوترها و اجرای آنها گشت. همچنین در اجرای همزمان برنامه توسط سیستم عامل، در اکثر اوقات برنامه‌های بار شده در حافظه شامل قطعه برنامه‌های کاملا مشابه و یکسانی بودند که به دفعات در حافظه کپی شده و فضای زیادی را هدر داده بودند، علاوه بر آن به ازای هر بار اجرای پروسه، تمام کد برنامه مجددا در حافظه کپی می‌شد.

بدین شکل ایده استفاده همزمان پروسه‌ها از یک نسخه کتابخانه بارگذاری شده در حافظه شکل گرفت. اهمیت این ایده زمانی بیشتر مشخص می‌گردد که بدانیم در حال حاضر تمام برنامه‌های مورد استفاده در سیستم عامل هر چند که کاملا با یکدیگر متفاوت باشند اما نیاز به بخشها و روتینهای زیادی دارند که در همه آنها مشترک است (روتینهای مورد استفاده در واسط گرافیکی، روتینهای مدیریت حافظه، کار با فایلها، ورودی و خروجی و ...).

پس با توجه به مطالب ذکر شده، لینک پویا بدین معنیست که زیربرنامه و روالهای کتابخانه مورد ارجاع، در زمان اجرا در حافظه بارگذاری خواهند شد نه در زمان کامپایل برنامه.

تنها کاری که در زمان کامپایل انجام می‌شود اینست که نام کتابخانه‌ها و آدرس توابع مورد استفاده آنها در برنامه ذخیره می‌گردد تا در زمان اجرا بتوان آنها را بارگذاری کرد، سپس عمل لینک توسط بخشی از هسته سیستم عامل به نام بارگذار (Loader) در زمان اجرا انجام می‌شود که وظیفه بارگذاری کد مورد نیاز در فضای پروسه برنامه را بر عهده دارد.

روش لینک پویا اولین بار در سال 1964 در نرم‌افزار (Michigan Terminal Systems) MTS مورد استفاده قرار گرفت، اما استفاده از آن به طور فراگیر در سال 1980 توسط شرکت Sun Micro Systems در سیستم عامل Unix SVR4 آغاز شد.

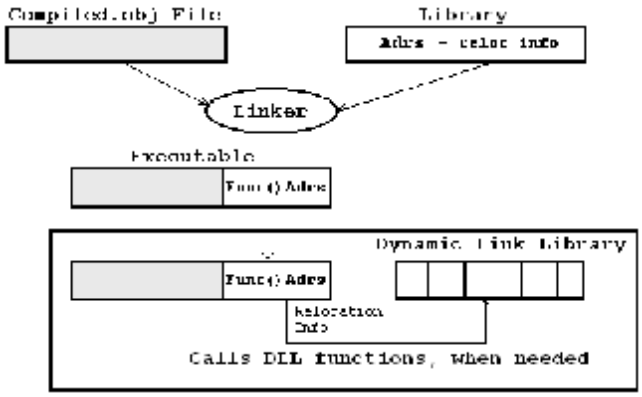
شرکت Sun با معرفی فرمت شئ (Executable and Linkable Format) ELF در این سیستم عامل، استفاده پویا از کتابخانه‌های اشتراکی را آغاز نمود. این فرمت به دلیل قابلیت انعطاف بالا و توسعه پذیری آن نسبت به دیگر فرمتهای معرفی شده قبلی، به عنوان استانداردی برای ساختار فایلهای اجرایی، کتابخانه‌های اشتراکی و Object فایلها در یونیکس و چند سیستم عامل دیگر انتخاب شد.

در حال حاضر نیز در سیستم عاملهای هم خانواده یونیکس مانند Linux, Solaris, IRIX, FreeBSD, NetBSD, OpenBSD, DragonFly BSD, Syllable و HP-UX مورد استفاده قرار می‌گیرد.

قابلیت انعطاف و توسعه این فرمت تا بدانجاست که امروزه در بسیاری از کنسولهای بازی مانند PSP, Wii, PlayStation2, PlayStation3, Nintendo DS و AmigaOS 4.0 به عنوان فرمت اجرایی برنامه‌ها مورد استفاده قرار می‌گیرد.

کتابخانه‌های اشتراکی ELF از قالب PICode یا کدهای مستقل از موقعیت (Position Independent Code) تبعیت کرده و در هر آدرسی از فضای حافظه پروسس می‌توانند بارگذاری شوند، چرا که حاوی تمام اطلاعات مورد نیاز جهت لینک در زمان اجرا مانند Global Offset Table بوده و فاقد هرگونه آدرسدهی مطلق می‌باشند.

در این قالب، کدها به نحوی کامپایل می‌شوند که آزادانه بتوانند در هر آدرس از فضای هر پروسسی بارگذاری شده، و یا اینکه تغییر مکان بدهند.



لینک پویا و ارجاع به توابع کتابخانه DLL در زمان اجرا

در حال حاضر عمده‌ترین استفاده کتابخانه‌های اشتراکی پویا در سیستم عامل ویندوز و فایلهای کتابخانه‌ای DLL (Dynaic link libraries) می‌باشد.

برخی از مزایای استفاده از کتابخانه‌های لینک پویا به شرح زیر است:

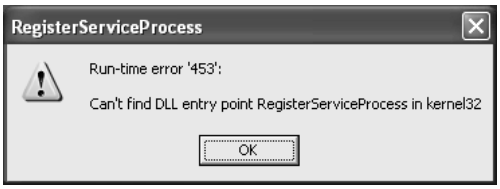
- با استفاده از این روش، حجم برنامه کامپایل شده خیلی کمتر شده و نیازی به الحاق روتینهای مورد ارجاع به فایل اجرایی نیست.
- با اجرای برنامه فضای کمتری از حافظه اصلی اشغال شده و در مصرف آن صرفه‌جویی می‌گردد.
- هر برنامه می‌تواند به شکل کاملا پویا در حین اجرا توابع و روتینهای مورد نیاز خود را از کتابخانه‌های مورد نظر بارگذاری و اجرا نماید.
- با استفاده مشترک همه برنامه‌ها از یک کتابخانه، به سادگی می‌توان کتابخانه‌های جدید را جایگزین نسخه‌های قدیمی آنها کرد (Library versioning)، بدون اینکه نیازی به کامپایل مجدد برنامه‌هایی باشد که از آن کتابخانه‌ها استفاده می‌کنند.

از معایب آنها نیز می‌توان به موارد زیر اشاره نمود:

- برنامه‌هایی که از این کتابخانه‌ها استفاده می‌کنند، برای اجرا به این کتابخانه‌ها نیاز دارند و به عبارتی قابلیت حمل این برنامه‌ها وابسته به این مسئله است که تمام کتابخانه‌های مورد استفاده در سیستم مقصد موجود باشند یا خیر. البته با فراگیر و استاندارد شدن کتابخانه‌های DLL و موجود بودن آنها در نسخه‌های مختلف سیستم عامل ویندوز، این مشکل تا حد زیادی کمرنگ شده است.

اختلال در اجرای برنامه به دلیل فقدان کتابخانه مورد نیاز

- قابلیت بالای این کتابخانه‌ها در Versioning گاهی باعث بروز مشکلاتی مانند عدم تطابق روتینهای یک کتابخانه با روتینهای مورد نیاز یک برنامه یا تغییر ساختار برخی روتینها در نسخه‌های جدید کتابخانه و بروز مشکلات ناهمخوانی با برنامه‌های قدیمی می‌گردد.



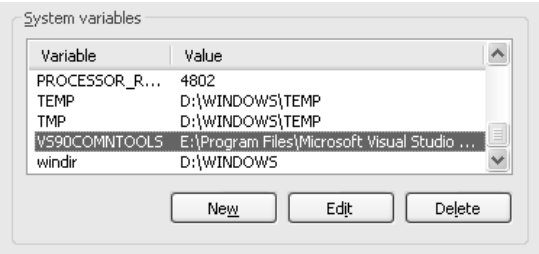
عدم وجود تابع RegisterServiceProcess در نسخه‌های جدید کتابخانه kernel32

- از دیگر مشکلات بسیار مهم این کتابخانه‌ها نقصهای امنیتی آنهاست که در بخشهای بعدی تشریح خواهند شد.

لینکهای پویا از قانونی به نام لینک بلادرنگ (Just-In-Time) یا JIT linking پیروی می‌کنند که بدین معنیست که برنامه‌ها دقیقا به محض اجرا به دنبال کتابخانه‌های مورد نیاز خود گشته و به آنها رجوع می‌کنند. لینکرها با قرار دادن نام و مسیر کتابخانه در فایل اجرایی، این مسئله را حل می‌نمایند. البته در صورتی که مسیر ثابتی برای کتابخانه مشخص نشده باشد، برای کشف مکان آن در سیستم عاملهای مختلف روشهای پیش فرضی وجود دارد.

روشی که در سیستم عامل ویندوز مورد استفاده قرار می‌گیرد به ترتیب زیر است:

- دایرکتوری حاوی برنامه، مورد جستجو قرار می‌گیرد.
- مسیری که توسط تابع SetDllDirectory() از توابع API در برنامه تعریف شده مورد جستجو قرار می‌گیرد.
- دایرکتوریهای System32 و System موجود در مسیر نصب سیستم عامل ویندوز (%Windir%) و پس از آنها خود دایرکتوری نصب ویندوز مورد بررسی قرار می‌گیرند.
- کلیه مسیرهای ثبت شده در متغیرهای محیطی سیستم عامل ویندوز (Environment Variables) مورد بررسی قرار می‌گیرند. این مسیرها توسط دستورات Set و Path در کنسول Dos Prompt قابل مشاهده بوده و توسط خود دستور Path یا بخش Startup And Recovery قابل تغییر هستند.



اعمال تغییرات در بخش Startup And Recovery

- بخشهایی از رجیستری (Registry) سیستم عامل ویندوز جهت تشخیص مسیرهای ممکن احتمالی مورد بررسی قرار می‌گیرند.

در صورتی که در هیچکدام از مراحل فوق، بارگذار سیستم عامل موفق به پیدا کردن کتابخانه مورد نظر نشود، روند اجرای برنامه مختل و متوقف شده و بارگذار پیامی مبنی بر عدم وجود کتابخانه صادر می‌نمید.

روند کاری موفق فایلهای کتابخانه‌ای DLL در سیستم عامل ویندوز مدیون قالب و ساختار خاص مربوط به فایلهای اجرایی در این سیستم عامل می‌باشد که Portable Executable یا PE نام دارد.

شرکت مایکروسافت در زمان عرضه ویندوز NT، فرمت PE را معرفی نمود. فایل PE به بلاکهایی به نام Section تقسیم شده که حاوی داده‌هایی با خصلتهای مختلف مثل Data/Code/Read/Write هستند. مثلاً سکشن text. حاوی کد اجرایی کامپایل شده برنامه بوده و یا سکشن rsrc. حاوی ریسورسهای مورد استفاده برنامه مثل تصاویر، اصوات یا فرمها است. در رابطه با روند کاری فایلهای DLL سکشن idata. که حاوی جدول آدرسهای نسبی توابع import شده به برنامه، یا به قولی جدول تمامی ارجاعات برنامه در فراخوانی توابع و روتینهای کتابخانه‌ای پویاست، مورد توجه قرار می‌گیرد..

پس از اجرای یک فایل PE، سیستم عامل این جدول را درون حافظه بارگذاری کرده و این آدرسها را به آدرس صحیح توابع مذکور در حافظه تغییر می‌دهد. دلیل وجود این جدول این است که فایلهای اجرایی همیشه در مکان ثابتی از حافظه بارگذاری نمی‌شوند.

.text	Code Section
CODE	Code Section of file linked by Borland Delphi or Borland Pascal
.data	Data Section
DATA	Data Section of file linked by Borland Delphi or Borland Pascal
.bss	Uninitialized data
.rdata	Section for Read-only Data
.idata	Import Table
.edata	Export Table
.tls	TLS Table
.reloc	Relocation Information
.rsrc	Resource Information

سکشنهای پیش‌فرض موجود در فایل PE

فایلها اجرایی ویندوز و DLL ها همگی دارای فرمت PE هستند. به توابع و روتینها موجود در کتابخانه‌های اشتراکی DLL، توابع API (Application Programming Interface) گفته می‌شود. در حال حاضر اساس کار سیستم عامل ویندوز بر مبنای این توابع استوار بوده و هر عملی با اجرای چند تابع API به انجام می‌رسد. برای استفاده از توابع API موجود در کتابخانه‌های DLL از دو روش اعلان ایستا و پویا استفاده می‌شود.

اعلان ایستا:

در این روش برنامه نویس ارجاع به روتین مورد نظر از کتابخانه را صریحا در کد برنامه ذکر می‌کند. با اینکار، در زمان شروع بارگذاری برنامه برای اجرا، بارگذار سیستم عامل ارجاعات مذکور را بررسی کرده و تمام روتینهای مورد نیاز را در فضای پروسه بارگذاری می‌نماید.

```
function BlockInput(..parameters..): DWORD; stdcall;
external 'user32.DLL';

Var  hresult:Byte ;
begin
    hresult := BlockInput (..parameters..);
end;
```

اعلان ایستا در زبان دلفی

اعلان پویا:

در این روش ارجاعات مورد نظر به شکل غیر صریح بوده و توسط توابع مربوط به بارگذاری کتابخانه‌ها در زمان اجرا (Runtime) انجام می‌شوند. برای اینکار از سه تابع LoadLibrary، FreeLibrary و GetProcAddress استفاده می‌شود.

با استفاده از تابع LoadLibrary کتابخانه DLL بررسی و هندل آن جهت استفاده در GetProcAddress مورد استفاده قرار می‌گیرد که توسط این تابع نیز می‌توان آدرس مکان روتین مورد ارجاع را در آن کتابخانه بدست آورد. پس از استفاده از روتین بارگذاری شده نیز، با استفاده از تابع FreeLibrary کتابخانه از حافظه آزاد می‌گردد. به عنوان مثال استفاده از توابع مذکور در زبان Delphi بدین شکل صورت می‌گیرد:

```
Var  SidRoutine:SidHandle;    hresult:Byte ;
begin
    hLib := LoadLibrary('advapi32.dll');
    @SidRoutine := GetProcAddress(hLib,
'ConvertSidToStringSidA');
    hresult := ConvertSidToStringSidA (..parameters..);
    FreeLibrary(hLib);
end;
```

اعلان پویا در زبان دلفی

ساختار DLL:

کتابخانه‌های DLL نیز خود نوعی فایل PE به شمار می‌آیند، چرا که مانند برنامه‌های عادی در محیط‌های برنامه نویسی ساخته شده و تنها تفاوت آنها استفاده از برخی مدلهای خاص مدیریت حافظه و ساخت روتینهایی جهت Import و Export توابع آنها می‌باشد.

به غیر از موارد ذکر شده این فایلها همانند دیگر برنامه‌ها در محیطهای برنامه نویسی کامپایل شده و به فایل کتابخانه‌ای با ساختار PE تبدیل می‌شوند. به عنوان مثال برای ساخت یک کتابخانه اشتراکی در زبان Delphi به شکل زیر عمل می‌شود:

```
library HLib;
type

function myProc(code : Integer; wParam, lParam :
LongInt) : LongInt; stdcall;

//          ...
var  mk : Thandle;
implementation
//          ...

procedure myProc(code : Integer; wParam, lParam :
LongInt) : LongInt;
begin
    //... Process received parameters ...
    result := CallNextHx (h, Code, wParam, lParam);
end;

function install (...):...; export;
begin
    mk := SetWinHx (hType, myProc, hInstance, 0);
end;

function Uninstall (...):...; export;
begin
    FreeAndNil(mk);
```

```
end;

exports install, Uninstall;

begin
//... dll assignments ...
end;
```

تعریف یک کتابخانه DLL در زبان دلفی

ضعف امنیتی فایلهای DLL:

با وجود تمام مزایای ذکر شده در روش لینک پویای کتابخانه‌های اشتراکی، برخی از این مزایا، خود باعث بوجود آمدن نقطه ضعفهای بزرگی در امنیت سیستم عامل می‌شوند.

مثلا به دلیل پویا و مستقل بودن این فایلها، هر برنامه مخربی می‌تواند اقدام به تخریب یا تغییر کتابخانه‌های مهم اشتراکی سیستم عامل نموده و بدین شکل صدمه بزرگی به تمام برنامه‌های ارجاع داده شده به آن کتابخانه وارد می‌گردد.

به عنوان مثال ویروسهای معروفی مانند Jeefo و Bagle از طریق تزریق خود به فایلهای اجرایی PE و یا تغییر در کتابخانه‌های اشتراکی DLL، به نوعی کد مخرب خود را در روند لینک پویای کتابخانه‌های اشتراکی قرار داده و از این طریق در مدت زمان بسیار کمی در فضای پروسس اکثر برنامه‌های سیستم عامل ویندوز بازنویسی می‌شوند.

همچنین یک ویروس می‌تواند آدرسهای مورد ارجاع لینک پویا در یک کتابخانه اشتراکی را به آدرس کدهای مخرب خود تغییر (Redirect) داده و روند اجرایی خود را تثبیت نماید.

شناسایی ویروس Bagle در سکشنهای PE

مثال شناخته شده دیگر در این زمینه، روش تزریق کد به حافظه پروسه‌ها توسط برنامه‌های مخرب، با استفاده از کلید AppInit_DLLs در رجیستری است. تمام DLL هایی که نام آنها در این کلید (مسیر) رجیستری ذکر شده باشد، به عنوان بخشی از کتابخانه User32.dll محسوب شده و به ساده‌ترین شکل ممکن با استفاده از تابع LoadLibrary توسط خود سیستم عامل، به درون حافظه تمام پروسه‌های فعال در حافظه بارگذاری می‌گردند.

اگر برنامه مخربی از این روش برای تزریق کد خود به حافظه پروسه‌ها استفاده کرده باشد، سربار حافظه بسیار بزرگی را برای تمام پروسسهای سیستم بوجود آورده و باعث اشغال حافظه سیستم و کند شدن روند پردازش در سیستم عامل می‌شود.

البته دلیل وجود این کلید رجیستری و مکانیسم مربوط به آن، استفاده از این قابلیت در برنامه‌هایست که نیاز به بارگذاری سراسری کدهای خود در حافظه پروسسها دارند، فلذا از این روش معمولا در نرم‌افزارهای سیستمی مانند دیباگرها یا برنامه‌های مانیتورینگ سیستم (مثل Google Desktop) استفاده می‌شود.

روند بارگذاری کتابخانه‌های پویا در حافظه:

همانطور که قبلا هم گفته شد وظیفه اصلی در این زمینه بر عهده بخش بارگذار (Loader) سیستم عامل است که بخشی از هسته سیستم عامل (Kernel) محسوب می‌گردد. Loader در هنگام بارگذاری خود سیستم عامل، اجرا شده و مقیم در حافظه می‌گردد و از آن پس اجرای هر برنامه دیگر را برعهده می‌گیرد. برای اجرای یک برنامه و بارگذاری آن در حافظه، روند زیر طی می‌شود:

- بارگذار فضای آدرسی مجازی (Virtual Address Space) برای برنامه ایجاد و رزرو می‌نماید.
- ماژولهای برنامه را از روی حافظه جانبی (دیسک) خوانده و آنها را در فضای آدرس رزرو شده بارگذاری و نگاشت (MAP) می‌کند.
- بارگذار با بررسی جدول سکشنها، آدرسهای حقیقی را از روی آدرسهای مجازی نسبی (RVA) محاسبه می‌نماید.
- سکشنها را یکی یکی و با توجه به آدرسهای محاسبه شده، در فضای پروسس رزرو شده بارگذاری می‌نماید.
- خصوصیات صفحات حافظه با توجه به نوع و مشخصات سکشنها تنظیم می‌گردند. مثلا سکشن text. دارای خصیصه فقط خواندنی و اجرایی (Execute/Readonly) بوده در صورتیکه سکشن data. دارای خصوصیات خواندن و نوشتن و غیر اجرایی (No-execute/ReadWrite) می‌باشد.
- در صورتی که آدرس پایه پیش‌فرض در سکشنها تعیین شده باشد، آدرس محاسبه شده و عمل جابه‌جایی (Relocation) انجام می‌گیرد.
- جدول ارجاعات به کتابخانه‌ها بررسی شده و تمام کتابخانه‌های مورد ارجاع یکی یکی جستجو شده و در فضای پروسس بارگذاری می‌شوند.
- جدول روتینهای خروجی (Export table) تمامی کتابخانه‌های اشتراکی بارگذاری شده، یکی یکی بررسی شده و آدرس روتینهای مورد ارجاع پیدا می‌شوند.
- آدرس تمام روتینهای مورد ارجاع در فراخوانیهای برنامه، موجود در جدول ارجاعات برنامه، ویرایش شده و با آدرس روتینهای بارگذاری شده در حافظه جایگزین می‌گردند.
- کنترل اجرا از بارگذار گرفته شده و به مدخل اجرایی برنامه (Program entrypoint) سپرده می‌شود.

مراجع:

- “Linkers and loaders”, John R. Levine I.E.C.C, 1999
- “Windows hooks”, Mohammad Shams Javi, 2007
- “Microsoft Portable Executable File Format Review”, Mohammad Shams Javi, 2008
- “Expert C programming”, Peter Van Der Linden
- “Dynamic-Link Library Search Order”, Microsoft Developer Network Library, 2003
- “Static Linking VS. Dynamic Linking”, Haitao Wang, Xiaomin Liu, 2008
- “Advantages of Using DLLs”, Microsoft Developer Network Library, 2003
- “Linking an Executable to a DLL”, Microsoft Developer Network Library, 2003
- “An In-Depth Look into the Win32 PE file format”, Matt Pietrek, 2002
- “PE Technical Reference”, Bernd Luevelsmeyer