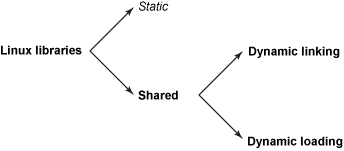
**Anatomia das Bibliotecas Dinâmicas do Linux**

*Processo e API*

As bibliotecas compartilhadas e vinculadas dinamicamente são um aspecto importante do GNU/Linux. ®. Elas permitem que executáveis acessem dinamicamente funcionalidades externas no momento da execução, reduzindo assim a área de cobertura geral de memória (trazendo funcionalidade quando necessário). Este artigo investiga o processo de criação e uso de bibliotecas dinâmicas, fornece detalhes sobre as diversas ferramentas para explorá-las e explora como tais bibliotecas funcionam na realidade.

As bibliotecas foram criadas para fornecer funcionalidade semelhante em uma única unidade. Essas unidades puderam então ser compartilhadas com outros desenvolvedores, permitindo o que veio a ser chamado de **programação modular** ou seja, criar programas de módulos. O Linux oferece suporte a dois tipos de bibliotecas, cada uma com suas próprias vantagens e desvantagens. A **biblioteca estática** contém funcionalidade ligada, de forma estática, a um programa no momento da compilação. Isso a difere das **bibliotecas dinâmicas,** que são carregadas quando um aplicativo é carregado e a ligação ocorre no momento da execução. A Figura 1 mostra a hierarquia de bibliotecas no Linux.

**Figura 1. Hierarquia de Bibliotecas no Linux**

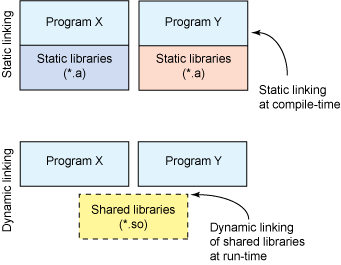


É possível usar as bibliotecas compartilhadas de várias formas: vinculadas dinamicamente no momento da execução ou carregadas dinamicamente e usadas no controle de programas. Este artigo explora esse dois métodos.

As bibliotecas estáticas podem ser úteis em pequenos programas, que exigem funcionalidade mínima. Para programas que exigem várias bibliotecas, as bibliotecas compartilhadas podem reduzir a área de cobertura de memória do programa (no disco e na memória no momento da execução). Isso ocorre porque vários programas podem usar uma biblioteca compartilhada simultaneamente; precisando então de apenas uma cópia da biblioteca na memória por vez. Com uma biblioteca estática, cada programa de execução tem sua própria cópia da biblioteca.

GNU/Linux fornece duas formas de lidar com as bibliotecas compartilhadas (cada método originado da Sun Solaris). É possível vincular dinamicamente seu programa com a biblioteca compartilhada, fazendo com que o Linux carregue a biblioteca na execução (a menos que ela já esteja na memória). Uma alternativa é fazer com que o programa chame funções seletivamente com a biblioteca em um processo chamado **carregamento dinâmico.** Com o carregamento dinâmico, um programa pode carregar uma biblioteca específica (a menos que já tenha carregado) e depois chamar uma determinada função dentro daquela biblioteca. (A Figura 2 mostra esses dois métodos.) Este é um padrão de uso comum na criação de aplicativos que oferecem suporte a plug-ins. Essa Interface de Programação de Aplicativo (API) será explorada e demonstrada posteriormente no artigo.

**Figura 2. Vínculo Estático vs. Dinâmico**



**Vínculo Dinâmico com o Linux**

Agora vamos iniciar o processo de uso de bibliotecas compartilhadas e vinculadas dinamicamente no Linux. Quando os usuários iniciam um aplicativo, eles estão chamando uma imagem Executable and Linking Format (ELF). O kernel se inicia com o processo de carregamento da imagem ELF na memória virtual do espaço do usuário. O kernel observa uma seção ELF chamada .interp, que indica o vinculador dinâmico que será usado (/lib/ld-linux.so), mostrado na Lista 1. Isso se assemelha à definição de arquivos de script no UNIX® (#!/bin/sh): Apenas usado em um contexto diferente.

**Lista 1. Usando readelf para Mostrar Cabeçalhos de Programas**

mtj@camus:~/dl$ **readelf -l dl**

Cada tipo de arquivo Elf é EXEC (executável)

Ponto de entrada 0x8048618

Há 7 cabeçalhos de programas, iniciando no deslocamento 52

Cabeçalhos de Programas:

Type Offset VirtAddr PhysAddr FileSiz MemSiz Flg Align

PHDR 0x000034 0x08048034 0x08048034 0x000e0 0x000e0 R E 0x4

INTERP 0x000114 0x08048114 0x08048114 0x00013 0x00013 R 0x1

[solicitando intérprete de programa: /lib/ld-linux.so.2]

LOAD 0x000000 0x08048000 0x08048000 0x00958 0x00958 R E 0x1000

LOAD 0x000958 0x08049958 0x08049958 0x00120 0x00128 RW 0x1000

DYNAMIC 0x00096c 0x0804996c 0x0804996c 0x000d0 0x000d0 RW 0x4

NOTE 0x000128 0x08048128 0x08048128 0x00020 0x00020 R 0x4

GNU\_STACK 0x000000 0x00000000 0x00000000 0x00000 0x00000 RW 0x4

...

mtj@camus:~dl$

Observe que ld-linux.so é, por si só, uma biblioteca ELF compartilhada, mas é estaticamente compilada e não possui dependências de bibliotecas compartilhadas. Quando o vínculo dinâmico é necessário, o kernel autoinicializa o vinculador dinâmico (intérprete ELF), que se autoinicializa, e depois carrega os objetos compartilhados especificados (a menos que eles já estejam carregados). Ele executa então as relocações necessárias, inclusive dos objetos compartilhados que o objeto de destino compartilhado utiliza. A variável de ambiente LD\_LIBRARY\_PATH define onde procurar os objetos compartilhados disponíveis. Quando concluído, o controle é transferido novamente ao programa original para iniciar sua execução.

A relocação é manipulada por meio de um mecanismo indireto chamado *Global Offset Table* (GOT) e *Procedure Linkage Table* (PLT). Essas tabelas fornecem os endereços das funções externas e dados, que o ld-linux.so carrega durante o processo de relocação. Isso significa que o código que requer o mecanismo indireto (ou seja, usa as tabelas) não precisa de alterações: apenas as tabelas exigem ajustes. A relocação precisa ocorrer imediatamente no carregamento ou sempre que uma determinada função é necessária. (Veja mais sobre essa diferença em [Carregamento Dinâmico com o Linux](http://www.ibm.com/developerworks/br/library/l-dynamic-libraries/#dynamicloading).)

Quando as relocações são concluídas, o vinculador dinâmico permite que qualquer objeto compartilhado carregado execute o código de inicialização opcional. Essa funcionalidade permite que a biblioteca inicialize dados internos e os prepare para uso. Esse código é definido na seção .init da imagem ELF. Quando a biblioteca está descarregada, ela também pode chamar uma função de término (definida como a seção .fini na imagem). Quando as funções de inicialização são chamadas, o vinculador dinâmico abandona o controle da imagem original que está sendo carregada.

[Voltar para parte superior](http://www.ibm.com/developerworks/br/library/l-dynamic-libraries/#ibm-pcon)

**Carregamento Dinâmico com o Linux**

Em vez de carregar automaticamente o Linux e vincular bibliotecas a um determinado programa, é possível compartilhar esse controle com o próprio aplicativo. Neste caso, o processo é chamado de *carregamento dinâmico.* Com o carregamento dinâmico, o aplicativo pode especificar uma determinada biblioteca para carregar e depois usá-la como executável (ou seja, chamar as funções dentro dela). Mas, como descrevemos anteriormente, a biblioteca compartilhada usada para o carregamento dinâmico não é diferente de uma biblioteca compartilhada padrão (um objeto ELF compartilhado). De fato, o vincular dinâmico ld-linux permanece envolvido neste processo como o loader e intérprete ELF.

A API de Dynamic Loading (DL) existe para o carregamento dinâmico e permite que uma biblioteca compartilhada fique disponível para um programa de espaço de usuário. Embora pequena, a API fornece tudo o que é necessário, com um fluxo de trabalho intenso. A API completa é mostrada na Tabela 1.

**Tabela 1. A API Dl**

|  |  |
| --- | --- |
| **Função** | **Descrição** |
| **dlopen** | Torna um arquivo de objeto acessível a um programa |
| **dlsym** | Obtém o endereço de um símbolo dentro de um arquivo de objeto dlopen |
| **dlerror** | Retorna um erro em sequência do último erro ocorrido |
| **dlclose** | Fecha um arquivo de objeto |

O processo começa com uma chamada para dlopen, fornecendo o objeto de arquivo para acessar um modo. O resultado da chamada de dlopen é uma manipulação do objeto que será usado posteriormente. O argumento mode informa ao vinculador dinâmico quando executar as relocações. Há dois valores possíveis. O primeiro, RTLD\_NOW, indica que o vinculador dinâmico concluirá todas as relocações necessárias no momento da chamada de dlopen. O segundo, um modo alternativo, RTLD\_LAZY, pede para que as relocações sejam executadas apenas quando forem necessárias. Isso é feito internamente, redirecionando todos os pedidos que ainda serão realocados através do vinculador dinâmico. Assim, o vinculador dinâmico saberá, no momento do pedido, quando uma nova referência estará ocorrendo e a relocação ocorrerá normalmente. Chamadas subsequentes não exigem a repetição da relocação.

Duas outras opções de modos estão disponíveis, bitwise OUed no argumento mode. RTLD\_LOCAL indica que os símbolos do objeto compartilhado que estão sendo carregados não serão disponibilizados para o processamento de relocação por nenhum outro objeto. Se isso é o que deseja (por exemplo, que o objeto compartilhado possa chamar símbolos na imagem de processo original), use RTLD\_GLOBAL.

A função dlopen também resolve automaticamente dependências em bibliotecas compartilhadas. Assim, se abrir um objeto dependente de outras bibliotecas compartilhadas, ele as carrega automaticamente. A função retorna um identificador usado nas chamadas subsequentes para a API. O protótipo de dlopen é:

#include <dlfcn.h>

void \***dlopen**( const char \*file, int mode );

Com um identificador para o objeto ELF, é possível identificar endereços para símbolos dentro do objeto usando a chamada dlsym. Essa função obtém um nome de símbolo, como o nome de uma função contida dentro do objeto. O valor de retorno é um endereço resolvido para o símbolo dentro do objeto:

void \***dlsym**( void \*restrict handle, const char \*restrict name );

Se ocorrer um erro durante uma chamada com esta API, será possível usar a função dlerror para retornar uma cadeia legível que representa o erro. Esse função não possui argumentos e retorna uma cadeia quando um erro anterior ocorre ou retorna NULL quando não ocorre nenhum erro:

char \***dlerror**();

Finalmente, quando nenhuma chamada adicional para o objeto compartilhado for necessária, o aplicativo pode chamar dlclose para informar o sistema operacional de que as referências ao identificador e ao objeto não são mais necessárias. Isso é corretamente contado como referência para que vários usuários de um objeto compartilhado não entrem em conflito uns com os outros (ele permanece na memória pois há um usuário para ele). Todos os símbolos resolvidos por meio de dlsym para o objeto fechado não serão mais disponibilizados.

char \***dlclose**( void \*handle );

[Voltar para parte superior](http://www.ibm.com/developerworks/br/library/l-dynamic-libraries/#ibm-pcon)

**Exemplo de Carregamento Dinâmico**

Agora que você já viu a API, vamos observar um exemplo da API DL. Neste aplicativo, você basicamente implementa um shell que permite que o operador especifique uma biblioteca, uma função e um argumento. Ou seja, o usuário pode especificar uma biblioteca e chamar uma função arbitrária dentro dessa biblioteca (que não estava vinculada anteriormente com este aplicativo). Resolva a função dentro da biblioteca usando a API DL e depois a chama dentro do argumento definido pelo usuário (emitindo o resultado). O aplicativo completo é mostrado na Lista 2.

**Lista 2. Shell para Usar a API DL**

#include <stdio.h>

#include <dlfcn.h>

#include <string.h>

#define MAX\_STRING 80

void invoke\_method( char \*lib, char \*method, float argument )

{

void \*dl\_handle;

float (\*func)(float);

char \*error;

/\* Abrir o objeto compartilhado \*/

dl\_handle = **dlopen**( lib, RTLD\_LAZY );

if (!dl\_handle) {

printf( "!!! %s\n", dlerror() );

return;

}

/\* Resolver o símbolo (método) do objeto \*/

func = **dlsym**( dl\_handle, method );

error = **dlerror**();

if (error != NULL) {

printf( "!!! %s\n", error );

return;

}

/\* Chamar o método resolvido e imprimir o resultado \*/

printf(" %f\n", (\*func)(argument) );

/\* Fechar o objeto \*/

dlclose( **dl\_handle** );

return;

}

int main( int argc, char \*argv[] )

{

char line[MAX\_STRING+1];

char lib[MAX\_STRING+1];

char method[MAX\_STRING+1];

float argument;

while (1) {

printf("> ");

line[0]=0;

fgets( line, MAX\_STRING, stdin);

if (!strncmp(line, "bye", 3)) break;

sscanf( line, "%s %s %f", lib, method, &argument);

invoke\_method( lib, method, argument );

}

}

Para criar este aplicativo, use a seguinte linha de compilação com o GNU Compiler Collection (GCC). A opção -rdynamic é usada para informar o vinculador sobre a inclusão de todos os símbolos na tabela de símbolos dinâmicos (permitindo retrocessos de rastreios com o uso de dlopen). A -ldl indica que dllib deve ser vinculado a este programa.

gcc -rdynamic -o dl dl.c -ldl

Voltando à [Lista 2](http://www.ibm.com/developerworks/br/library/l-dynamic-libraries/#list2), a função main atua simplesmente como intérprete, analisando três argumentos da linha de entrada (nome da biblioteca, nome da função, argumento de ponto de flutuação). Se bye estiver presente, o aplicativo existirá. Caso contrário, os três argumentos serão analisados para a função invoke\_method, que usa a API DL.

Comece com uma chamada para dlopen para obter acesso ao arquivo de objeto. Se um identificador NULL for retornado, o objeto não poderá ser localizado e o processo será encerrado. Caso contrário, você terá um identificador para o objeto que poderá ser interrogado posteriormente. Usando a função de API dlsym, tente resolver o símbolo dentro do arquivo de objeto recentemente aberto. Será obtido um ponteiro válido para o símbolo ou NULL e retornará um erro.

Com o símbolo resolvido no objeto ELF, a próxima etapa é simplesmente chamar a função. Observe a diferença entre este código e a discussão anterior do vínculo dinâmico. Neste exemplo, você força o endereço do símbolo no arquivo de objeto para um ponteiro de função e o chama. O exemplo anterior usou o nome do objeto como uma função e o vinculador dinâmico garante que o símbolo aponte para o local correto. Embora o vinculador dinâmico possa fazer todo o "trabalho sujo", essa abordagem permite que sejam criados muitos aplicativos dinâmicos que podem ser estendidos no momento da execução.

Depois de chamar sua função de destino no objeto ELF, feche o acesso a ele por meio de uma chamada para dlclose.

Um exemplo de como usar este programa de teste é mostrado na Lista 3. Nesse exemplo, compila-se e depois executa-se o programa. Depois, chama-se algumas funções dentro da biblioteca correspondente (libm.so). Com base nesta demonstração, o programa pode chamar funções arbitrárias dentro de um objeto compartilhado (biblioteca) usando o carregamento dinâmico. Este é um recurso poderoso e permite a extensão de programas com novas funcionalidades.

**Lista 3. Usando o Programa Simples para Chamar Funções da Biblioteca**

mtj@camus:~/dl$ **gcc -rdynamic -o dl dl.c -ldl**

mtj@camus:~/dl$ **./dl**

> **libm.so cosf 0.0**

1.000000

> **libm.so sinf 0.0**

0.000000

> **libm.so tanf 1.0**

1.557408

> **bye**

mtj@camus:~/dl$

[Voltar para parte superior](http://www.ibm.com/developerworks/br/library/l-dynamic-libraries/#ibm-pcon)

**Ferramentas**

O Linux fornece uma variedade de ferramentas para visualizar e analisar objetos ELF (inclusive bibliotecas compartilhadas). Uma das mais úteis é o comando ldd, que é usado para emitir dependências de bibliotecas compartilhadas. Por exemplo, usar o comando ldd em seu aplicativo dl mostra o seguinte:

mtj@camus:~/dl$ **ldd dl**

linux-gate.so.1 => (0xffffe000)

libdl.so.2 => /lib/tls/i686/cmov/libdl.so.2 (0xb7fdb000)

libc.so.6 => /lib/tls/i686/cmov/libc.so.6 (0xb7eac000)

/lib/ld-linux.so.2 (0xb7fe7000)

mtj@camus:~/dl$

O que ldd está dizendo é que essa imagem ELF depende de linux-gate.so (um objeto compartilhado especial que manipula chamadas do sistema e não tem nenhum arquivo associado no sistema de arquivos), libdl.so (a API DL), a biblioteca GNU C (libc.so) e, finalmente, o carregador Linux dinâmico (pois há dependências de bibliotecas compartilhadas).

O comando readelf é um utilitário com muitos recursos que permite que analise e leia objetos ELF. Um uso interessando de readelf é identificar os itens que podem ser realocados dentro de um objeto. Em nosso programa simples (mostrados na [Lista 2](http://www.ibm.com/developerworks/br/library/l-dynamic-libraries/#list2)), é possível ver os símbolos que exigem relocação como:

mtj@camus:~/dl$ **readelf -r dl**

A seção de relocação '.rel.dyn' no deslocamento 0x520 contém 2 entradas:

Offset Info Type Sym.Value Sym. Name

08049a3c 00001806 R\_386\_GLOB\_DAT 00000000 \_\_gmon\_start\_\_

08049a78 00001405 R\_386\_COPY 08049a78 stdin

A seção de relocação '.rel.plt' no deslocamento 0x530 contém 8 entradas:

Offset Info Type Sym.Value Sym. Name

08049a4c 00000207 R\_386\_JUMP\_SLOT 00000000 dlsym

08049a50 00000607 R\_386\_JUMP\_SLOT 00000000 fgets

08049a54 00000b07 R\_386\_JUMP\_SLOT 00000000 dlerror

08049a58 00000c07 R\_386\_JUMP\_SLOT 00000000 \_\_libc\_start\_main

08049a5c 00000e07 R\_386\_JUMP\_SLOT 00000000 printf

08049a60 00001007 R\_386\_JUMP\_SLOT 00000000 dlclose

08049a64 00001107 R\_386\_JUMP\_SLOT 00000000 sscanf

08049a68 00001907 R\_386\_JUMP\_SLOT 00000000 dlopen

mtj@camus:~/dl$

Desta lista, é possível ver as diversas chamadas de bibliotecas C que exigem relocação (para libc.so), incluindo chamadas para a API DL (libdl.so). A função \_\_libc\_start\_main é uma função de biblioteca C chamada antes da função main do seu programa (um shell que fornece inicialização necessária).

Outros utilitários que operam em arquivos de objetos incluem objdump, que exibe informações sobre arquivos de objetos e nm, que lista os símbolos dos arquivos de objetos (inclusive informações sobre depuração). Também é possível chamar o vinculador dinâmico do Linux diretamente com o programa ELF como seu argumento para iniciar manualmente a imagem:

mtj@camus:~/dl$ **/lib/ld-linux.so.2 ./dl**

> **libm.so expf 0.0**

1.000000

>

Adicionalmente, é possível usar ld-linux.so para listar as dependências de uma imagem ELF (de forma idêntica ao comando ldd) usando a opção --list. Lembre-se de que ele é apenas um programa de espaço de usuário autoinicializado pelo kernel quando necessário.

[Voltar para parte superior](http://www.ibm.com/developerworks/br/library/l-dynamic-libraries/#ibm-pcon)

**Indo Além**

Este artigo esboçou os fundamentos de alguns dos recursos do vinculador dinâmico. Na zona Linux do [Recursos](http://www.ibm.com/developerworks/br/library/l-dynamic-libraries/#resources) abaixo, serão encontradas introduções mais detalhadas para o formato de imagens ELF e a relocação de símbolos e processos. E, ainda, como sempre no Linux, será possível fazer download da fonte do vinculador dinâmico (consulte [Recursos](http://www.ibm.com/developerworks/br/library/l-dynamic-libraries/#resources)) para entender suas estruturas internas.

**Recursos**

**Aprender**

* No artigo de Peter Seebach "[Dissecting shared libraries](http://www.ibm.com/developerworks/web/library/l-shlibs.html)" (developerWorks, janeiro de 2005), aprenda sobre as bibliotecas compartilhadas, como criá-las e as diversas ferramentas disponíveis para operá-las.
* [SkyFree.org](http://www.skyfree.org/) fornece uma ótima [introdução ao ELF](http://www.skyfree.org/linux/references/ELF_Format.pdf) (PDF), abrangendo arquivos de objetos, carregamento de programas e a biblioteca C. A Wikipedia também fornece uma [breve descrição sobre o ELF](http://en.wikipedia.org/wiki/Executable_and_Linkable_Format) e muitos links para recursos adicionais para o ELF (especificações e interfaces para muitas arquiteturas do processador).
* O blog [EM\_386 de Chris Rohf](http://em386.blogspot.com/2006/10/resolving-elf-relocation-name-symbols.html) fornece uma descrição detalhada sobre a resolução de símbolos ELF e todos os seus detalhes adicionais. Ele explica as tabelas GOT e PLT e suas manipulações por meio do vinculador dinâmico do Linux.
* A Wikipedia possui boas fontes sobre as bibliotecas [e](http://en.wikipedia.org/wiki/Library_%28computing%29) [bibliotecas estáticas](http://en.wikipedia.org/wiki/Static_library). É possível obter informações sobre [vinculadores](http://en.wikipedia.org/wiki/Linker) e [carregadores](http://en.wikipedia.org/wiki/Loader_%28computing%29) e seus relacionamentos com as bibliotecas.
* Para uma ótima introdução ao ELF, leia "[Standards and specs: An unsung hero: The hardworking ELF](http://www.ibm.com/developerworks/power/library/pa-spec12/index.html)" (developerWorks, dezembro de 2005). O ELF é o formato de objeto padrão para Linux. Ele é um formato de arquivo flexível que abrange imagens executáveis, objetos, bibliotecas compartilhadas e até mesmo core dumps. Encontre informações mais detalhadas nesta [referência de formato](http://www.skyfree.org/linux/references/ELF_Format.pdf) e [documento de especificação do ELF](http://refspecs.linux-foundation.org/LSB_3.2.0/LSB-Core-generic/LSB-Core-generic/elf-generic.html).
* O artigo de *Linux Journal*, "[Linkers and Loaders](http://www.linuxjournal.com/article/6463)" (novembro de 2002) oferece uma ótima introdução aos objetivos além dos ligadores e carregados usando arquivos ELF (inclusive resolução e relocação de símbolos).
* Na zona Linux do [developerWorks](http://www.ibm.com/developerworks/linux/), encontre mais recursos para desenvolvedores Linux e varra nossos [artigos e tutoriais mais conhecidos](http://www.ibm.com/developerworks/linux/library/l-top-10.html).
* Consulte todas as [dicas de Linux](http://www.ibm.com/developerworks/views/linux/libraryview.jsp?topic_by=All+topics+and+related+products&sort_order=desc&lcl_sort_order=desc&search_by=linux+tip%3A&search_flag=true&type_by=All+Types&show_abstract=true&start_no=1&sort_by=Date&end_no=100&show_all=false) e [tutoriais de Linux](http://www.ibm.com/developerworks/views/linux/libraryview.jsp?topic_by=All+topics+and+related+products&sort_order=desc&lcl_sort_order=desc&search_by=&search_flag=&type_by=Tutorials&show_abstract=true&sort_by=Date&end_no=100&show_all=false) no developerWorks.
* Fique atualizado com [eventos técnicos e Webcasts do developerWorks](http://www.ibm.com/developerworks/offers/techbriefings/).