Insper

Bibliotecas: lincagem estática e dinâmica

Igor S. Montagner

Parte 1 - examinando arquivos executáveis

Baixe o arquivo atividade_lincagem.zip do blackboard. O arquivo deve conter duas pastas (`bin` e `src`). Antes de prosseguir examine o código na pasta `src`. Os arquivos *bin/exemplo1* e *bin/exemplo2* são duas compilações do arquivo *main.c* usando opções diferentes.

Abra o arquivo *bin/exemplo1* no gdb (mas não o execute) e responda às seguintes questões.

1. Quais são as funções definidas pelo usuário?

2. Quais funções são chamadas no `main()`?

3. Estas funções foram listadas no item 1?

4. Qual a saída de *exemplo1*? **Você pode executar o programa**

Agora abra o arquivo *bin/exemplo2* no gdb (mas não o execute) e responda.

1. Quais são as funções definidas pelo usuário?

2. É possível identificar quais funções são chamadas no `main()`?

3. Tente executar *bin/exemplo2*. Qual a saída do programa?

Examine então o arquivo *bin/libDynamic.so* usando o *gdb*.

1. Quais as funções definidas neste arquivo?

2. Qual a relação de *exemplo2* com *libDynamic.so*?

3. Pesquise como usar a variável *LD_LIBRARY_PATH* para executar *bin/exemplo2*.

4. Por que é necessário usar esta variável de ambiente para executar *bin/exemplo2*?

5. Qual é a saída de *bin/exemplo2*? **Aqui você pode rodar o programa**

6. Pesquise qual é o diretório padrão para bibliotecas em Linux. Seu código funciona se a biblioteca *libDynamic.so* estiver copiada neste diretório?

Parte 2 - Bibliotecas

Bibliotecas são conjuntos de funções agrupadas com algum propósito específico. Podemos ter bibliotecas para Entrada/Saída, processamento de imagens, computação gráfica, etc. De uma certa maneira, bibliotecas são como Lego: construímos um programa combinando blocos definidos em bibliotecas com código específico para resolver nosso problema.

Exemplo: uma biblioteca de Entrada/Saída contém várias funções como `printf, scanf, fprintf, scanf`, etc. A biblioteca só *define* estas funções. Cada função as usa da maneira que for desejada.

Exemplo: uma biblioteca de processamento de imagens pode disponibilizar diversas operações básicas de imagens. Cada programa *utiliza* essas operações de uma maneira diferente, atingindo resultados completamente diferentes.

No nível mais baixo uma biblioteca precisa possuir

1. Uma interface de programação - API (quais funções existem, quais parâmetros cada função recebe, etc)
2. Implementação para cada função disponibilizada.

- Pesquise para que servem arquivos *header* `.h` em *C*.

- Com base na sua resposa acima, como você os utilizaria para definir a API de uma biblioteca?

Uma biblioteca coloca a **implementação** das funções declaradas nos arquivos `.h` em um (ou mais) arquivos `.c`. Cada arquivo é compilado individualmente em um arquivo objeto `.o`. A compilação de um arquivo individual é feita pelo comando

```
$ gcc -c arquivo.c
```

Este comando gera `arquivo.o`. Este arquivo não pode ser usado sozinho, mas podemos combinar vários arquivos `.o` em um executável (se algum deles definir a função `main`) ou uma biblioteca (se temos um conjunto de funções auxiliares mas não um `main`). No caso de bibliotecas, como elas são usadas em outros programas dois problemas são centrais:

- **Distribuição**: entregar sempre os fontes pode não ser nem prático (para quem usa) nem desejável (para quem distribui). Por isso, precisamos de formatos binários para bibliotecas da mesma maneira que temos formatos executáveis (`ELF` em Linux, `.exe/PE` para Windows).
- **Definição da API**: como exportar as funções disponibilizadas na biblioteca? Neste caso é suficiente distribuir os arquivos *header* `.h`.

Portanto, toda bibliteca é formada por duas partes: arquivos header `.h` que definem sua *API* e um arquivo binário que contém a implementação das funções exportadas nos `.h`.

Existem duas soluções para o problema da distribuição de bibliotecas: estática e dinâmica. Uma biblioteca estática (*.a* no linux, *.lib* no Windows) é copiada para dentro do executável final e fica misturada com o código do programa, assim como vimos no arquivo *bin/exemplo1*. Uma biblioteca dinâmica é armazenada em um arquivo separado (*.so* no linux, *.dll* no Windows) que é carregado em memória no momento da execução do programa. Vários programas podem carregar a mesma biblioteca dinâmica ao mesmo tempo e podem existir diversas versões da mesma biblioteca em um sistema. Essa técnica foi usada no arquivo *bin/exemplo2*.

Vamos ver agora como usar o *gcc* com bibliotecas. Pesquise e responda:

- O gcc compila código em 4 etapas (pré-processamento, compilação, montagem e lincagem). Descreva o que é feito em cada uma delas e apresente o comando que a executa.

- Quais argumentos são usados para especificar **onde** procurar por bibliotecas? E por cabeçalhos?

- Como especificamos quais bibliotecas um programa usa? O gcc irá procurar nos caminhos determinados no item acima e nos caminhos padrão do sistema.

- Pesquise sobre o comando `pkg-config` e como ele pode ser usado para facilitar a utilização de bibliotecas em *C/C++*.

Parte 3 - lincagem estática

Uma biblioteca estática nada mais é que um conjunto de arquivos objeto mais informações de quais funções estão presentes e quais variáveis globais são usadas. Sua criação é feita usando o comando `ar`.

- Quais opções são usadas para criar uma biblioteca estática? Explique o que cada opção usada faz e digite um exemplo completo para criar a *libStatic.a* a partir dos arquivos *lib1.c* e *lib2.c*.

- Digite abaixo o comando para compilar *main.c* usando a biblioteca criada no passo acima. Você pode supor que ambos arquivos estão no mesmo diretório.

Parte 4 - lincagem dinâmica

Diferentemente de uma biblioteca estática, em que o código da biblioteca é incorporado ao executável, uma biblioteca dinâmica (ou compartilhada) é copiada para a memória no momento da execução. O arquivo executável contém pedidos para o sistema carregar a biblioteca e devolver, em lugares pré-estabelecidos, os endereços das funções desejadas. Logo, dependemos da presença de um arquivo *.so* contendo as implementações das funções que usamos em locais pré-estabelecidos do sistema.

1. O que faz a variável *LD_LIBRARY_PATH*?

2. Como podemos adicionar um caminho de maneira permanente na lista de bibliotecas?

Como o código pode ser carregado em qualquer lugar da memória precisamos compilá-lo com a opção `-fPIC`, que gera *código independente de posição*. Qualquer referência interna da biblioteca é feita usando endereçamento *relativo*. Como não existem endereços absolutos no código, é possível copiá-lo em qualquer lugar da memória que ele funcionará corretamente!

- Qual a sequência de comandos do gcc para compilar uma biblioteca compartilhada? Seu exemplo deve criar uma biblioteca *libDynamic.so* a partir dos arquivos *lib1.c* e *lib2.c*

- Digite abaixo o comando para compilar *main.c* usando a biblioteca criada no passo acima. Você pode supor que ambos arquivos estão no mesmo diretório.

Parte 5 - juntando tudo

1. Em qual momento as funções de uma biblioteca estática são incorporadas ao executável? E para uma biblioteca dinâmica?

2. Cite uma vantagem e uma desvantagem da utilização de bibliotecas estáticas.

3. Cite uma vantagem e uma desvantagem da utilização de bibliotecas dinâmicas.

Parte 6 - Make (para casa)

A ferramenta *Make* é usada para automatizar a compilação de executáveis.

- Pesquise como criar um *Makefile* básico para fazer a compilação das bibliotecas compiladas nos exercícios anteriores e produzir dois executáveis a partir do *main.c*:
  1. *main_static* deverá contar a biblioteca compilada estaticamente e rodar "direto"
  2. *main_dynamic* deverá compilar usando a biblioteca dinâmica e rodar somente se ela estiver no `LD_LIBRARY_PATH`
- Crie um *makefile* para compilar os arquivos lib1.c e lib2.c em uma biblioteca estática chamada *libStatic.a* e em uma biblioteca dinâmica chamada *libDynamic.so*. Crie um target para compilar o *main.c* usando a versão dinâmica (executável main_dynamic) e um usando a versão estática (executável main_static). Rode ambos e verifique a saída. Você deve entregar o roteiro preenchido mais os arquivo de projeto incluindo seu *Makefile*.