Conceitos básicos de programação de sistemas

ambientes UNIX e Linux

Taisy Silva Weber

conceitos básicos

programação de sistemas

- ✓ programação em ambientes UNIX:
  - executáveis e scripts
  - compiladores C e C++,
  - arquivos *header*,
  - bibliotecas estáticas e compartilhadas,
  - desenvolvimento modular de programas
    - usando arquivos *header* e bibliotecas

programação em UNIX

- ✓ programação em C

  considerada linguagem padrão para programação de sistemas

  - 1969 - primeira versão de Unix escrita em Assembler para o PDP 7
  - 1973 - kernel reescrito em C (Ritchie e Thompson)

- ✓ atualmente:
  - vasta gama de linguagens disponíveis
    - veremos **C** e programação usando **shell (sh)**

programas Unix

- ✓ aplicações em arquivos de dois tipos
  - executáveis & scripts
- ✓ executáveis
  - rodam diretamente

semelhantes a `.exec` do DOS

- ✓ scripts
  - conjunto de instruções (diretivas) para outro programa (um interpretador)

correspondem a `.bat` no DOS

executáveis e scripts

- ✓ nenhuma diferença entre eles para o usuário
  - podem ser substituídos um pelo outro
- ✓ Unix não usa extensões
  - `.exec` e `.bat` **não** tem significado em Unix
  - programas são localizados em arquivos com o mesmo nome
  - variable da shell: `path`

shell: programa interpretador de linhas de comandos

localização padrão

- ✓ *paths* onde programas do sistema são encontrados
  - paths são determinados pelo administrador do sistema
  - `/bin`
  - `/usr/bin`
  - `/usr/local/bin`

programas para uso geral supridos pelo sistema

usar hierarquia `/usr/local` para nossos programas

programas instalados pelo administrador para uma máquina específica

compilador C

- ✓ `c89`
- ✓ `cc`
- ✓ `gcc`
- ✓ experimentar auxílio:
  - `man gcc`
  - `info gcc`

em Linux todos esses comandos se referem ao compilador **GNU C**

tipicamente localizado em `/usr/bin` ou `/usr/local/bin`

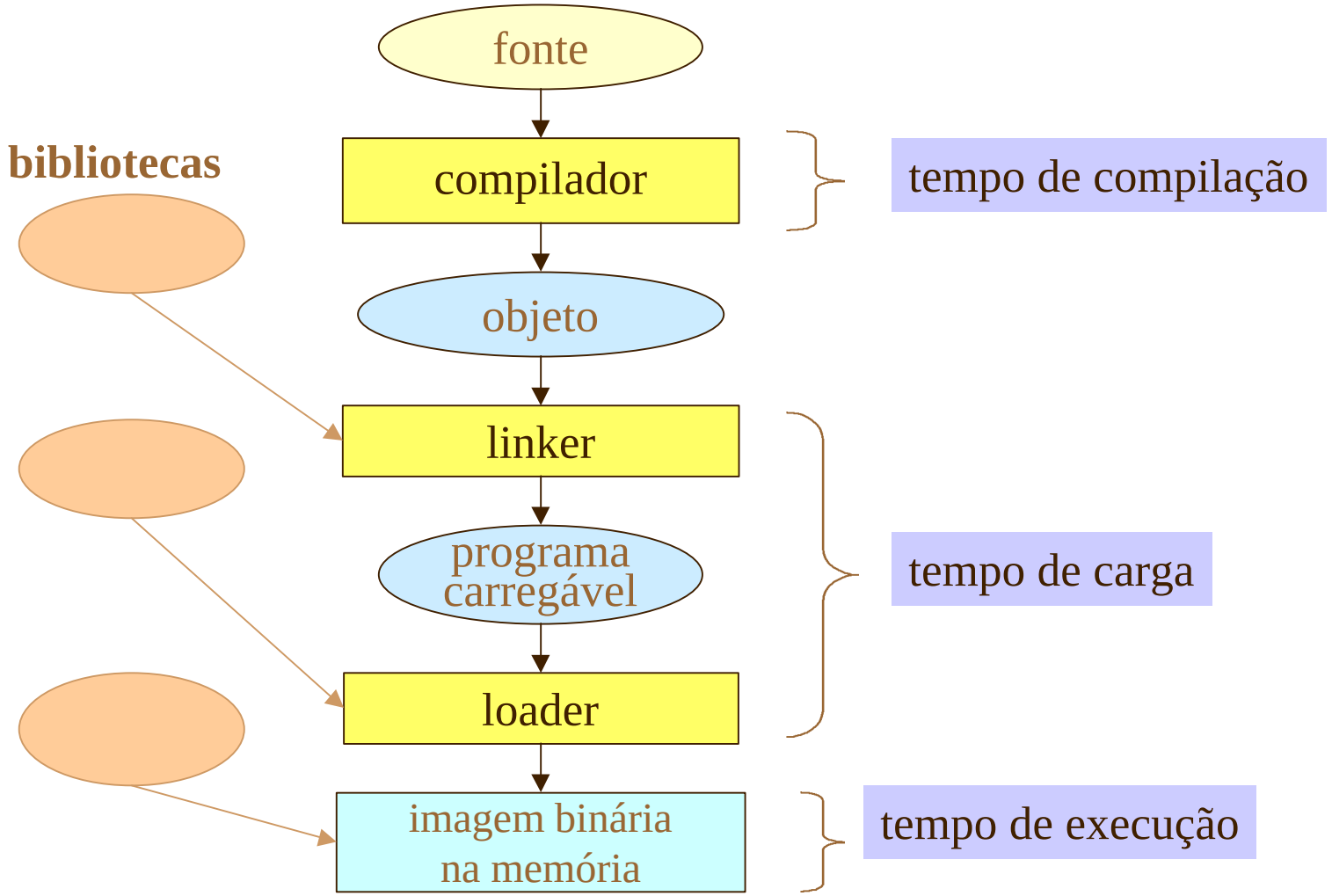
compila e linka

online manual pages

lab

versão completa de documentação do compilador com centenas de páginas

compilador, linker e loader



gcc: opções

- ✓ opções em linha de comando — lembrar de Textinfo
  - serão estudados os mais importantes
  - -o *nomedoarq* — especifica nome do arquivo de saída
  - -c — compila sem linkar
  - -I*dir* — outro *dir* para procurar por *include files*
  - -l*foo* — linka com lib*foo*

gcc: extensões

- algumas extensões não seguem padrão ANSI

✓ duas áreas particularmente úteis

- interface com código em linguagem Assembler

  conferir em:

  http://www.rt66.com/~brennan/djgpp/djgpp_asm.html

- construção de bibliotecas compartilhadas
  - particularmente importante devido a filosofia de reuso, foco e modularidade
  - *header* files

primeiro programa

```
#include <stdio.h>

int main()
{
        printf(”Alô mundo\n”);
        exit(0);
}
```

editar o programa usando um editor

compilar, linkar e executar

```
$ cc -o alo alo.c
$ ./alo
Alô mundo
$
```

lab

arquivos header...

- provêm:
  - declaração de constantes
  - declarações para chamadas de sistema ou chamadas de funções em bibliotecas
  - localização para C:
  - `/usr/include`
    - `/usr/include/sys`
    - `/usr/include/linux`
  - para C++
    - `/usr/include/g++-2`

compilador localiza automaticamente

...arquivos header...

- ✓ diretórios não padrões
  - flag **`-I`** para C:

-I*dir* : outro *dir* para procurar por *include files*

```
$ gcc -I/usr/openwin/include prog.c
```

o compilador é dirigido para localizar no diretório definido (*dir*) além dos diretórios padrões

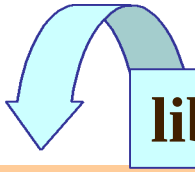
- diretório definido por -I: `/usr/openwin/include`
- válido para *headers* incluídos no programa `prog.c`

bibliotecas

- ✓ *libraries* **lib**
  - coleção de funções pré-compiladas
    - tipicamente funções relacionadas
      - GUI library
      - dbm library

exemplos

  - reuso
  - localização padrão
    - `/lib`
    - `/usr/lib`



lib não padrão

O *linker* precisa ser avisado da sua localização. O nome da **lib** deve seguir uma **convenção** e deve ser mencionado na **linha de comando** do compilador.

nome para biblioteca

- sempre começa com `lib`
- continua com identificação da lib
  - `libc` C library
  - `libm` mat library

  } exemplos
- última parte especifica tipo da lib
  - `.a` para libs estáticas
  - `.so` e `.sa` para libs compartilhadas

usualmente libs existem nos dois formatos

por default é usada a lib compartilhada

verificar: `ls /usr/lib`

lab

localização das libs...

o compilador sempre procura na biblioteca **C** padrão

compilação e linkagem

✓ instruindo o compilador

- fornecendo o nome completo (full path name)

```
$ cc -o fred fred.c /usr/lib/libm.a
```

procura adicionalmente no arquivo especificado

- usando flag -l

```
$ cc -o fred fred.c -lm
```

abreviatura para **libm** em uma das libs padrão. O compilador escolherá a lib compartilhada se ela existir.

localização das libs

compilação e linkagem

- ✓ instruindo o compilador
  - – localização não padrão
    - usando flag -L

flag -l

flag -L

```
$ cc -o Xfred -L/usr/openwin/lib Xfred.c -lX11
```

abreviatura para **libX11**

diretório onde se encontra a **libX11**

bibliotecas estáticas e bibliotecas compartilhadas

- programas podem obter funções de bibliotecas de duas formas:

static libraries

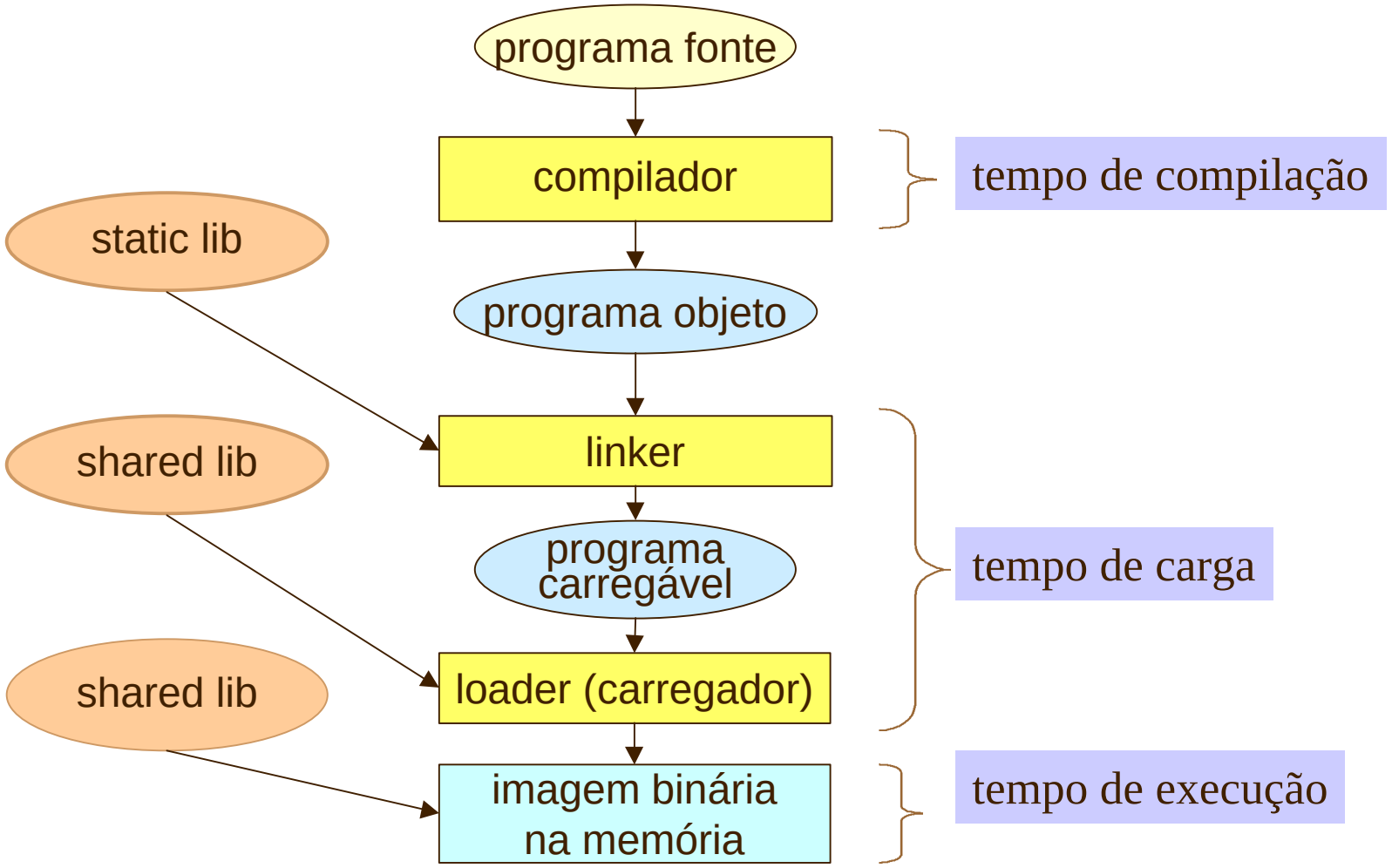
- funções são **copiadas** de uma biblioteca estática diretamente **no código** executável

shared libraries

- funções são indiretamente **referenciadas** em uma biblioteca compartilhada e lidas apenas quando o executável roda

tb. **shared object** ou **dynamically linked library**

bibliotecas e a geração de código



lib estática

também conhecida como **archive**

- ✓ simples **coleção de arquivos** `.a`
  - funções (ou rotinas) prontas para uso
  - quando um programa precisar de uma função da biblioteca, deve-se incluir um *header* que declara a função
  - compilador e *linker* combinam o programa e a **lib** em um único código executável
    - não esquecer de usar opção `-l` para indicar outras libs além da lib C padrão

criação de lib estática

- usar programa `ar` (archive)
- compilar fontes separadamente com `cc -c`
  - se possível tentar manter cada função em um arquivo fonte separado

cria `.o` (sem linkar)

```
ar rcs libname.a foo.o bar.o baz.o
```

```
ar rcs libname.a foo.o
ar rcs libname.a bar.o
ar rcs libname.a baz.o
```

r: replace
c: create
s: symbol table
v: verbose

`lab`

programa `ar`

- programa `ar` (*archive*)
  - cria um "arquivo" e insere arquivos objetos nele
    - no caso *archive* significa coleção de arquivos individuais colocados juntos em um grande arquivo
    - o programa `ar` serve para criar um arquivo contendo arquivos de qualquer tipo, não apenas objetos
  - `ranlib` cria um índice para o *archive*
    - não é necessário para Linux

      exigido em alguns sistemas derivados do BSD

desvantagens da lib estática

- ✓ replicação inútil de código
  - cada programa executável contém sua própria cópia das rotinas da biblioteca
- – grande consumo de memória
  - tanto MP com disco
- – tempo perdido na carga de funções que já estão na memória sendo usadas por outros programas

uma solução para esses problemas é usar lib compartilhada

lib compartilhada

existe apenas uma cópia do código da função na memória

- ✓ compartilhamento de funções
  - todos os programas na MP que usam uma função, usam o **mesmo código** dessa função na MP
  - – economia de:
    - – espaço de memória
    - – tempo de carga de uma cópia para cada programa
  - – desvantagem: maior complexidade
    - executável é formado por um conjunto de partes independentes
      - – se o usuário não tem uma das libs, o programa não roda

lib compartilhada

- também é um conjunto de arquivos objeto
  - código de cada objeto da lib deve ser independente de posição (PIC)
  - programas linkados a uma shared lib não contém o código da lib

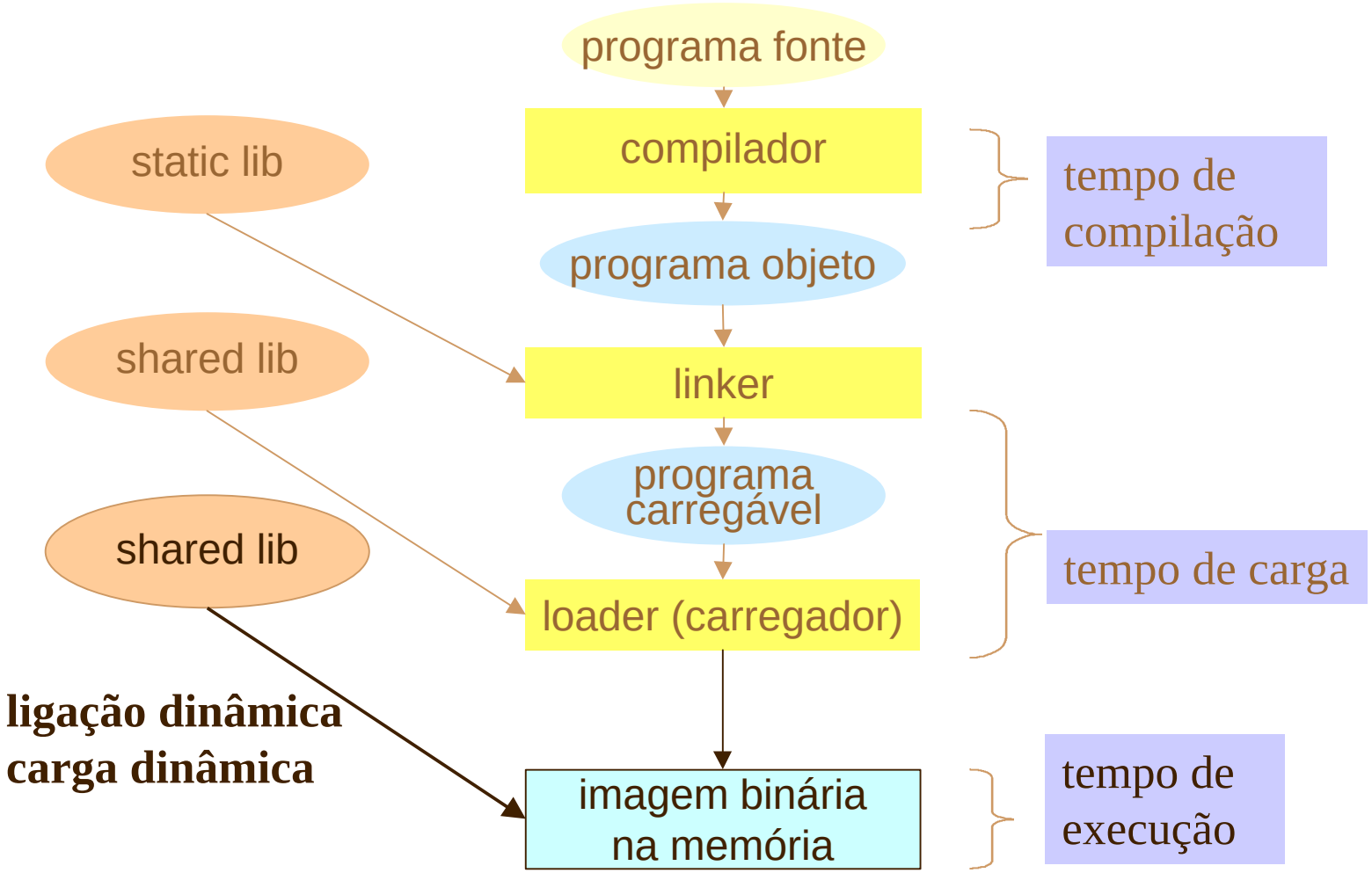
basta uma cópia do objeto na memória para todos os processos

stub

incluído no código para cada chamada de rotina da biblioteca

indica como localizar a rotina na memória ou como carregá-la se ainda não estiver na memória

carga & ligação dinâmica



ligação dinâmica

✓ ligação dinâmica

dynamic link

basta uma cópia da rotina na memória para todos os processos

apoio do **SO** para compartilhar rotinas por vários processos

linking é postergado até o momento de execução

shared lib

carga dinâmica

✓ carga dinâmica

sem necessidade de apoio do **SO**

dynamic load

programa que chama deve gerenciar carga da rotina

REF: Silberschatz cap 9

uma rotina só é carregada na memória quando for chamada

pode ser **shared lib** ou outro **shared object** qualquer

rotinas são mantidas em disco no formato PIC

uma rotina que não for chamada não ocupa espaço na MP

lib compartilhada em Linux

REF: Johnson cap 7

- originalmente
  - usada uma forma binária simplista
  - criação e manutenção difícil e complicada
- atualmente
  - formato **ELF** (executable and linking format)
    - válido para praticamente qualquer plataforma Linux
  - ainda mais complicado do que criar `.a`
    - algumas restrições
    - preservação de compatibilidade com antigas versões ou indicação de incompatibilidade

Linux soname

- soname - nome especial para shared lib
  - formado por **nome da lib** e **número de versão**
  - número de versão é usado para gerenciar compatibilidade
    - enquanto as libs tiverem o mesmo **soname**, elas são consideradas compatíveis

exemplo

soname: libc.so.5
biblioteca: libc.so.5.2.15
(`ldconfig` cria um link simbólico do soname para o nome real)

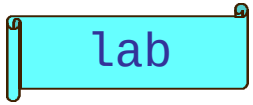
uma nova versão com **soname** libc.so.6 não seria considerada compatível para programas que usam libc.so.5

Criação de shared lib

- analisar problemas de compatibilidade
- gerar código independente de posição
  - opção -fPIC no gcc
- cuidar para não inibir ação de depuradores
- para linkar usar gcc e não ld
- não esquecer o **soname**
  - opção -Wl
- e outras regrinhas discutidas no livro texto (Johnson cap 7)

utilitário `ldd`

- `ldd`
  - mostra as *shared libs* que um programa usa (ao qual um programa está linkado)
  - `$ ldd nomedoprograma`
  - as libs listadas precisam estar disponíveis quando o programa executa
- `ldd` irá listar uma lib chamada `ld-linux.so`



equivalências

item	UNIX	DOS
programa objeto	funcao.o	FUNCAO.OBJ
executável	progr	PROG.EXE
lib estática	bib.a	BIB.LIB
lib compartilhada	bib.so	arquivos .DLL
	bib.sa	~ arquivos .LIB

desenvolvimento modular de programas

- ✓ estilo de programação
  - UNIX encoraja programadores a desenvolver programa com sua filosofia

simplicidade, foco, reuso, filtros, formatos simples, flexibilidade

- ✓ aula prática
  - desenvolvimento modular criando bibliotecas estáticas
    - ênfase em simplicidade e reuso
    - ref Stones pgs 13, 15, 20 e 21