Ligador, carregador, DLL

Ligador: fundamentos

- **Se programa tratado como bloco único**
  - **Alteração em uma linha de código**
  - **Requereria total recompilação/remontagem**
    - » **Ineficiente**
  - **Limitaria o uso de bibliotecas**
- **Solução pragmática:**
  - **Dividir o programa em módulos**
  - **Só recompilar/remontar módulo alterado**

Ligador: consequência da modularidade

- **É necessário um programa ligador**
  - **Emenda módulos que foram compilados/montados isoladamente**
    - » **“Linkeditor” (“linker”)**
- **Justificativa**
  - **É mais eficiente “remendar”**
    - » **Algumas linhas de código**
  - **Que recompilar/remontar**
    - » **Todo o código**

Ligador: etapas da ligação

- Posicionar código e dados em memória
  - Simbolicamente
    - » Pois feito em arquivo no disco
- Determinar endereços de referências
  - Labels de instruções e dados
- Editar referências
  - Internas
    - » Aquelas dentre as já resolvidas pelo montador, mas que precisem ser modificadas
  - Externas
    - » Não resolvidas

Ligador: edição de referências

- **Referências a serem editadas ocorrem em:**
  - **Desvios** (os que usam modo absoluto, i.e. não relativo ao PC)
    - » Referências a instruções
  - **Load/store** (as que usam modo absoluto ⇒ relativo a gp)
    - » Referências a dados
- **Para editar/resolver referências, usam-se:**
  - **Informações de relocação**
    - » Lista de referências anotadas que requerem edição
  - **Tabela de símbolos**
    - » Lista de referências não resolvidas

Ligador: pressupostos da montagem

- **Módulos montados independentemente**
- **Mesmo posicionamento na memória**
  - **Código no início do segmento `.text`**
  - **Dados no início do segmento `.data`**
- **Posição relativa dos módulos**
  - **Desconhecida em tempo de montagem**
  - **Determinada em tempo de ligação**
    - » **Relocação**

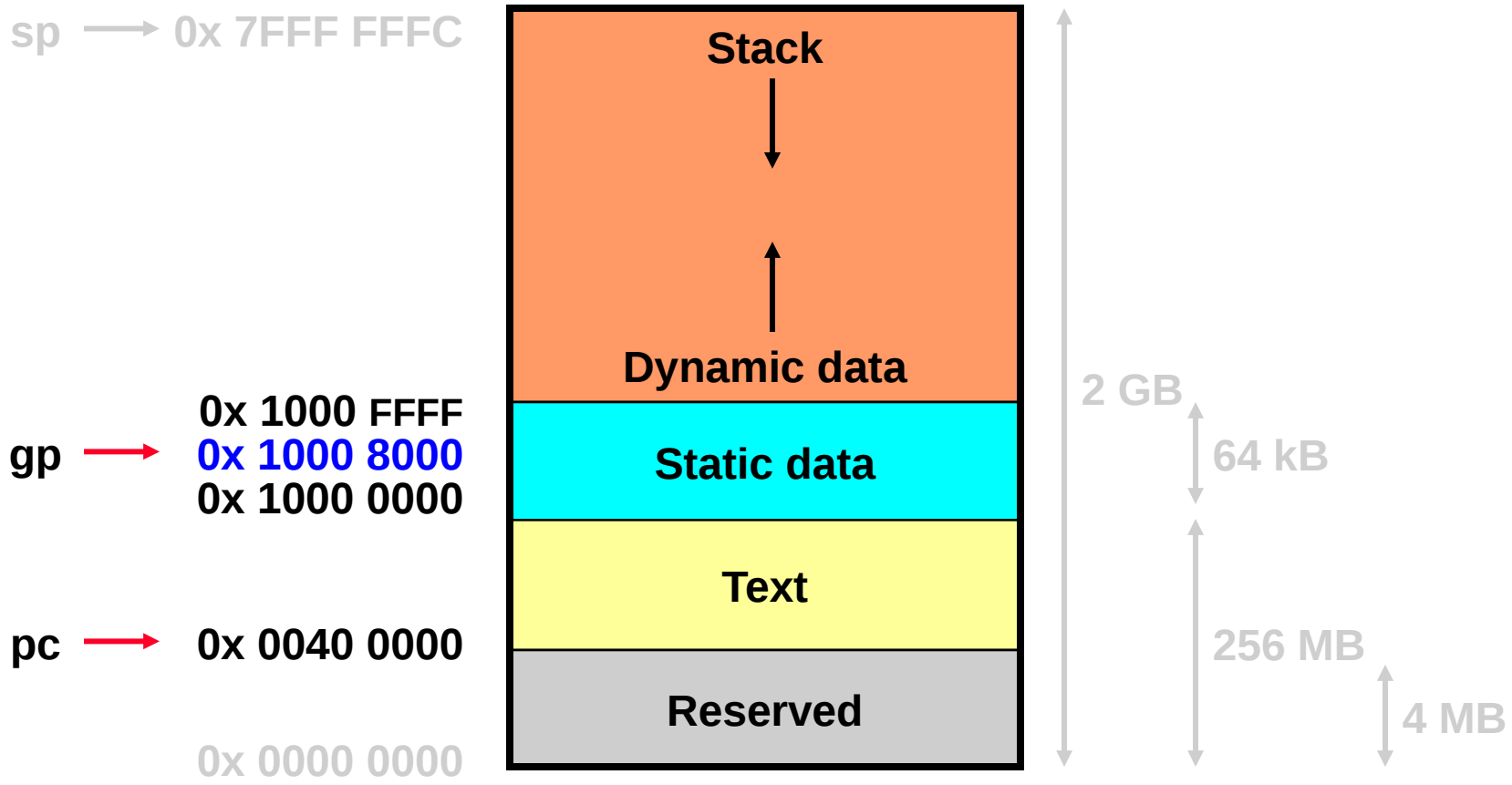
Ligador: relocação dos módulos

- **Cada módulo foi posicionado**
  - **Em seu próprio espaço de endereçamento**
  - **Referências absolutas resultam inválidas**
    - » **Precisam ser editadas**
    - » **Exemplo: jal 0x0004 → jal 0x0040 0004**
- **Relocação garante compatibilidade**
  - **Das referências usadas nos módulos**
  - **Com o novo espaço de endereçamento**

Ligador: resultado da ligação

- Ligador produz arquivo executável
- Formato similar ao do arquivo objeto
  - Mas sem referências indefinidas
  - Nem informação de relocação

Uso da memória: revisitando o layout



Ligador: exemplo

Object file header	Name	Procedure A	
	Text size	0x 100	
	Data size	0x 20	
Text segment	Address	Instruction	
	0	lw $a0, 0($gp)	
	4	jal 0	
	...	...	
Data segment	0	(X)	
	...	...	
Relocation info	Address	Instruction type	Dependency
	0	lw	**X**
	4	jal	**B**
Symbol table	Label	Address	
	X	???	
	B	???	

Código objeto do Procedimento A

Ligador: exemplo

Object file header	Name	Procedure B	
	Text size	0x 200	
	Data size	0x 30	
Text segment	Address	Instruction	
	0	sw $a1, 0($gp)	
	4	jal 0	
	...	...	
Data segment	0	(Y)	
	...	...	
Relocation info	Address	Instruction type	Dependency
	0	sw	**Y**
	4	jal	**A**
Symbol table	Label	Address	
	Y	???	
	A	???	

Código objeto do Procedimento B

Ligador: exemplo

Executable file header	Name		
	Text size	**0x 300**	
	Data size	**0x 50**	
Text segment	Address	Instruction	
	0	lw $a0, 0($gp)	**256 bytes**
	4	jal 0	
	…	…	
	0	sw $a1, 0($gp)	**512 bytes**
	4	jal 0	
	…	…	
Data segment	0	(X)	**32 bytes**
	…	…	
	0	(Y)	**48 bytes**
	…	…	

Código executável: atualização do cabeçalho

Ligador: exemplo

Executable file header	Name	
	Text size	0x 300
	Data size	0x 50
Text segment	Address	Instruction
	0x 0040 0000	lw $a0, 0($gp)
	0x 0040 0004	jal 0
	…	…
	0x 0040 0100	sw $a1, 0($gp)
	0x 0040 0104	jal 0
	…	…
Data segment	0	(X)
	…	…
	0	(Y)
	…	…

Código executável: relocação de código

Ligador: exemplo

Executable file header	Name	
	Text size	0x 300
	Data size	0x 50
Text segment	Address	Instruction
	0x 0040 0000	lw $a0, 0($gp)
	0x 0040 0004	jal 0
	...	...
	0x 0040 0100	sw $a1, 0($gp)
	0x 0040 0104	jal 0
	...	...
Data segment	**0x 1000 0000**	(X)
	...	...
	0x 1000 0020	(Y)
	...	...

Código executável: relocação de dados

Ligador: exemplo

Executable file header	Name	
	Text size	0x 300
	Data size	0x 50
Text segment	Address	Instruction
	0x 0040 0000	lw $a0, 0($gp)
	0x 0040 0004	jal **0x 40 0100**
	...	...
	0x 0040 0100	sw $a1, 0($gp)
	0x 0040 0104	jal **0x 40 0000**
	...	...
Data segment	0x 1000 0000	(X)
	...	...
	0x 1000 0020	(Y)
	...	...

Código executável: edição de referências em desvios

Ligador: exemplo

Executable file header	Name	
	Text size	0x 300
	Data size	0x 50
Text segment	Address	Instruction
	0x 0040 0000	lw $a0, **0x 8000**($gp)
	0x 0040 0004	jal 0x 40 0100
	...	...
	0x 0040 0100	sw $a1, **0x 8020**($gp)
	0x 0040 0104	jal 0x 40 0000
	...	...
Data segment	0x 1000 0000	(X)
	...	...
	0x 1000 0020	(Y)
	...	...

Código executável: edição de referências em load/store

0x 10008000 (base)
- 0x 10000000 (efetivo)
0x 00008000 (-offset)

0x 00008000
Trocar sinal
0x FFFF7FFF
+ 0x 00000001
0x FFFF8000 = 0x 8000

0x 10008000
+ 0x 8000

Executable file header	Name	
	Text size	0x 300
	Data size	0x 50
Text segment	Address	Instruction
	0x 0040 0000	lw $a0, 0x 8000($gp)
	0x 0040 0004	jal 0x 40 0100
	...	...
	0x 0040 0100	sw $a1, 0x 8020($gp)
	0x 0040 0104	jal 0x 40 0000
	...	...
Data segment	0x 1000 0000	(X)
	...	...
	0x 1000 0020	(Y)
	...	...

Código executável: edição de referências em load/store

0x 10008000 (base)
- 0x 10000000 (efetivo)
0x 00008000 (-offset)

0x 00008000
↓ Trocar sinal
0x FFFF7FFF
+ 0x 00000001
0x FFFF8000 = 0x 8000

0x 10008000 (base)
+ 0x FFFF8000 (offset)
0x 10000000 (efetivo)

Executable file header	Name	
	Text size	0x 300
	Data size	0x 50
Text segment	Address	Instruction
	0x 0040 0000	lw $a0, **0x 8000**($gp)
	0x 0040 0004	jal 0x 40 0100
	…	…
	0x 0040 0100	sw $a1, **0x 8020**($gp)
	0x 0040 0104	jal 0x 40 0000
	…	…
Data segment	0x 1000 0000	(X)
	…	…
	0x 1000 0020	(Y)
	…	…

Código executável: edição de referências em load/store

Carregador: fundamentos

- Arquivo executável reside no disco
  - Após a etapa de ligação
- Sistema operacional precisa:
  - Ler o arquivo executável no disco
  - Carregá-lo em memória
  - Disparar sua execução
- Componente do sistema operacional
  - O programa carregador (“loader”)

Carregador: passos de processamento

- Lê o cabeçalho do arquivo executável
  - Para determinar tamanho dos segmentos
    - » Texto e dados
- Cria um espaço de endereçamento
  - Suficiente para acomodar texto e dados
- Copia instruções e dados
  - Do arquivo executável para a memória

Carregador: passos de processamento

- Copia parâmetros do programa principal
  - Na pilha (se existirem parâmetros)
- Inicializa os registradores
  - Os de uso geral e o sp
- Inicia e termina a execução
  - Desvia para a rotina de inicialização
    - » Que copia os parâmetros para registradores
  - Chama a rotina principal
  - Em seu retorno, termina o programa
    - » Chamada de sistema (`syscall` exit)

DLL:
a noção de ligação dinâmica

- **Mecanismo estudado é ligação estática**
  - **Rotinas de bibliotecas são ligadas**
  - **Antes de o programa entrar em execução**
- **Desvantagem 1:**
  - **Não permite “upgrade” de bibliotecas**
    - » **Como rotinas estão incorporadas no executável**
    - » **Programa continua usando versão antiga**
    - » **Mesmo que versão nova disponível**

DLL:
a noção de ligação dinâmica

- **Desvantagem 2:**
  - **Todas as rotinas invocáveis são carregadas**
    - » **Mas somente as usadas no código**
  - **Mesmo que nem todas sejam usadas**
    - » **Em uma dada execução do programa**
  - **Ocupação ineficiente de espaço em memória**
    - » **Por exemplo: todas as rotinas da biblioteca C padrão usadas no código ⇒ 2,5 MB**

DLL:
a noção de ligação dinâmica

- **Desvantagem 3:**
  - **Possível redundância no sistema de arquivos**
    - » **Cópias de uma mesma rotina em executáveis distintos**
- **Desvantagem 4:**
  - **Possível redundância em memória**
    - » **Cópias de uma mesma rotina em executáveis distintos**

DLL:
a noção de ligação dinâmica

- **Bibliotecas ligadas dinamicamente**
  - **Rotinas de bibliotecas não ligadas/carregadas**
  - **Até programa ser carregado/entrar em execução**
    - » **“Dynamic Linked Libraries” (DLLs)**
- **Requisito:**
  - **Programas/bibliotecas mantêm informação extra**
    - » **Para localização de procedimentos não locais**

DLL:
alternativas de implementação

- **Alternativa 1:**

  Ligação dinâmica em tempo de carga

  - **Carregador invoca ligador**
    - » **Para encontrar bibliotecas apropriadas**
    - » **Resolver as referências externas**
- **Desvantagem:**
  - **Todas as rotinas invocáveis são ligadas**
    - » **Embora nem todas efetivamente invocadas em tempo de execução**

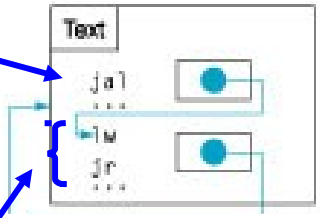
DLL: alternativas de implementação

- **Alternativa 2:**

  Ligação dinâmica em tempo de execução
  - **Cada rotina é ligada o mais tarde possível**
  - **Somente depois de ser invocada**
    - » **“Lazy procedure linkage”**
- **Vantagens:**
  - **Só rotinas efetivamente invocadas são carregadas**
  - **Desempenho próximo ao obtido com ligação estática**
    - » **Exceto na primeira invocação (degradação de desempenho)**

DLL: mecanismo

Chamadas a rotinas não locais (interceptadas por rotinas auxiliares)



Chamadas a rotinas auxiliares ao final do código (uma por rotina não local)

DLL: mecanismo

Chamadas a rotinas não locais (interceptadas por rotinas auxiliares)

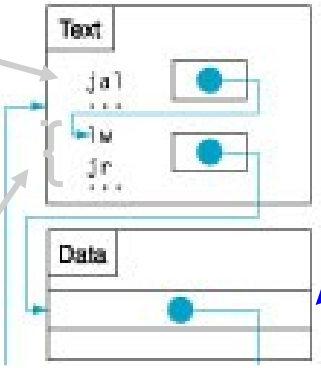


Tabela de endereços das rotinas não locais (uma entrada por rotina)

Chamadas a rotinas auxiliares ao final do código (uma por rotina não local)

DLL: mecanismo

Chamadas a rotinas não locais (interceptadas por rotinas auxiliares)

Chamadas a rotinas auxiliares ao final do código (uma por rotina não local)

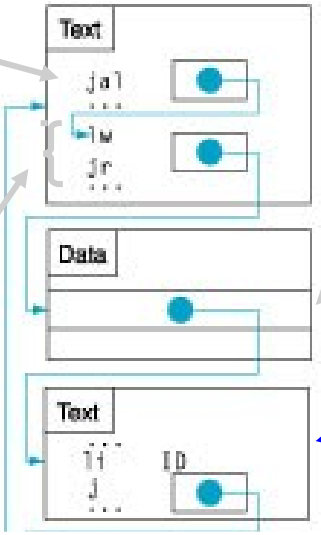


Tabela de endereços das rotinas não locais (uma entrada por rotina)

Código identificador das rotinas não locais (um ID por rotina)

DLL: mecanismo

Chamadas a rotinas não locais (interceptadas por rotinas auxiliares)

Chamadas a rotinas auxiliares ao final do código (uma por rotina não local)

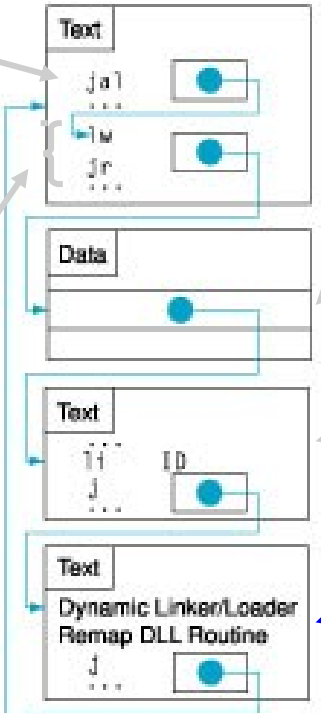


Tabela de endereços das rotinas não locais (uma entrada por rotina)

Código identificador das rotinas não locais (um ID por rotina)

Código do ligador/carregador (residente em memória)

DLL: mecanismo

Chamadas a rotinas não locais (interceptadas por rotinas auxiliares)

Chamadas a rotinas auxiliares ao final do código (uma por rotina não local)

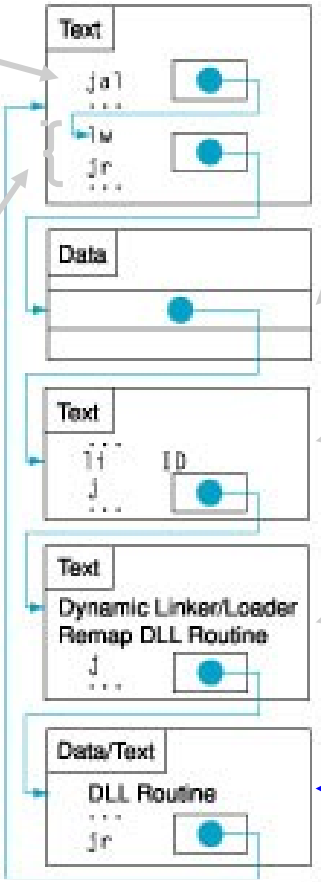


Tabela de endereços das rotinas não locais (uma entrada por rotina)

Código identificador das rotinas não locais (um ID por rotina)

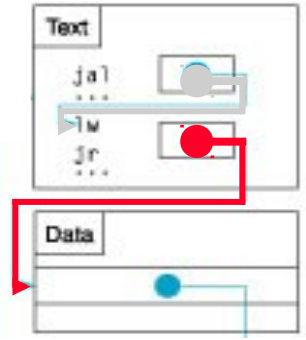
Código do ligador/carregador (residente em memória)

Código/dados da rotina não local (carregados em memória)

DLL:
primeira invocação da rotina

Passo 1: invocação da rotina não local

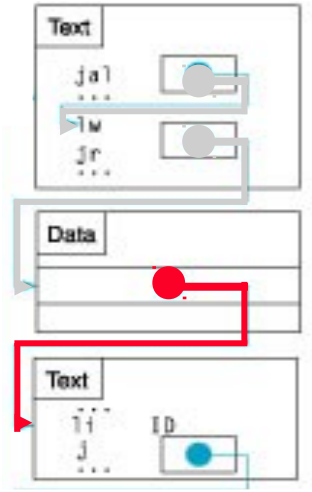
DLL:
primeira invocação da rotina



Passo 1: invocação da rotina não local

Passo 2: interceptação pela rotina auxiliar

DLL:
primeira invocação da rotina

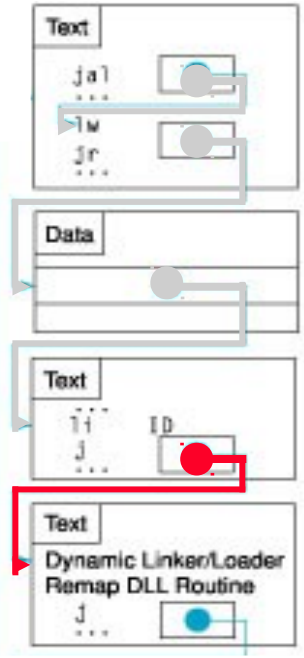


Passo 1: invocação da rotina não local

Passo 2: interceptação pela rotina auxiliar

Passo 3: identificação da rotina não local

DLL:
primeira invocação da rotina



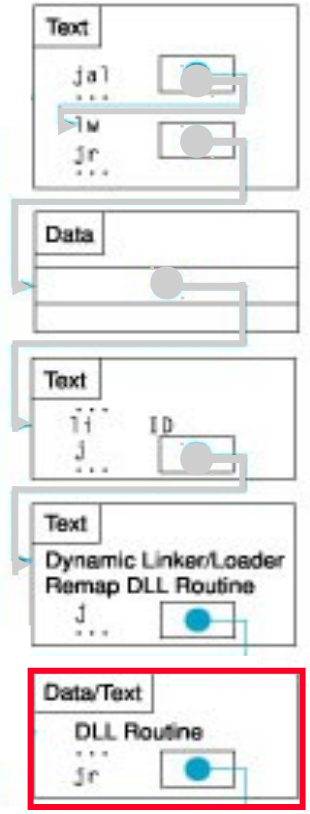
Passo 1: invocação da rotina não local

Passo 2: interceptação pela rotina auxiliar

Passo 3: identificação da rotina não local

Passo 4: desvio para o carregador/ligador

DLL: primeira invocação da rotina



Passo 1: invocação da rotina não local

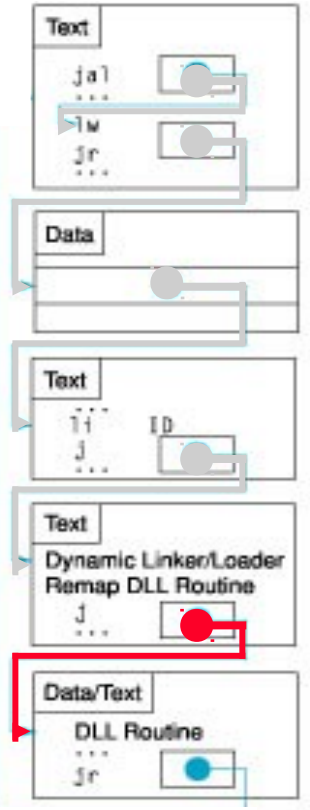
Passo 2: interceptação pela rotina auxiliar

Passo 3: identificação da rotina não local

Passo 4: desvio para o carregador/ligador

Passo 5: carga e ligação da rotina identificada

DLL: primeira invocação da rotina



Passo 1: invocação da rotina não local

Passo 2: interceptação pela rotina auxiliar

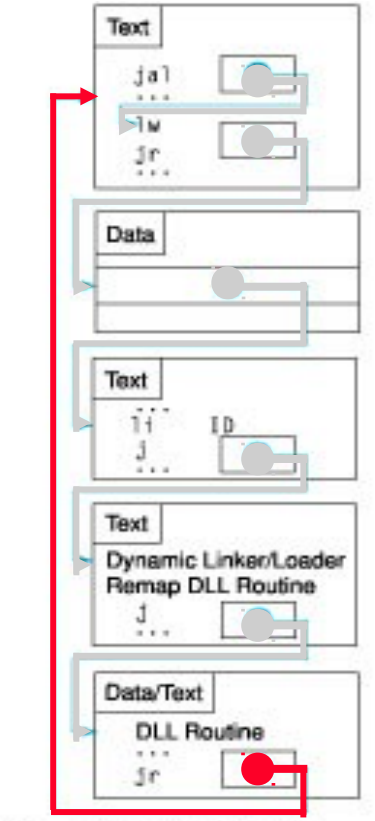
Passo 3: identificação da rotina não local

Passo 4: desvio para o carregador/ligador

Passo 5: carga e ligação da rotina identificada

Passo 6: desvio para a rotina carregada

DLL: primeira invocação da rotina



Passo 1: invocação da rotina não local

Passo 2: interceptação pela rotina auxiliar

Passo 3: identificação da rotina não local

Passo 4: desvio para o carregador/ligador

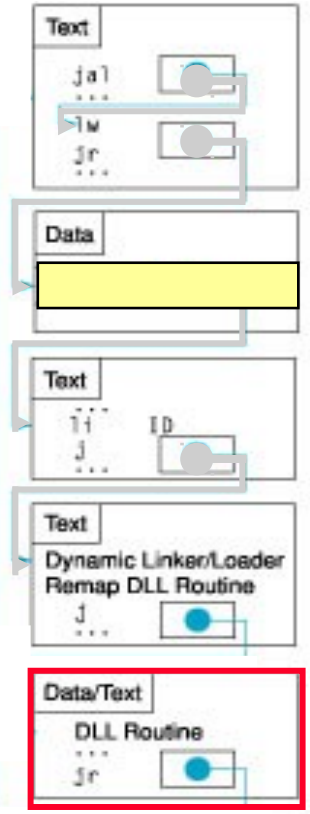
Passo 5: carga e ligação da rotina identificada

Passo 6: desvio para a rotina carregada

Passo 7: retorno da rotina executada

DLL: revisitando o Passo 5

Quando rotina é ligada, seu endereço em memória é atualizado na tabela de endereços



Passo 1: invocação da rotina não local

Passo 2: interceptação pela rotina auxiliar

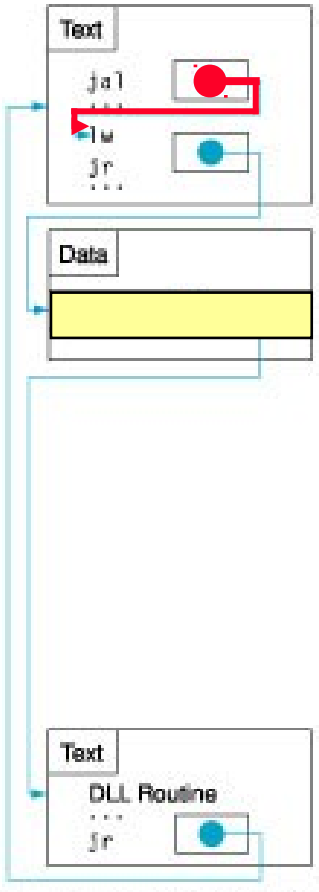
Passo 3: identificação da rotina não local

Passo 4: desvio para o carregador/ligador

Passo 5: carga e ligação da rotina identificada

DLL:
invocações subseqüentes da rotina

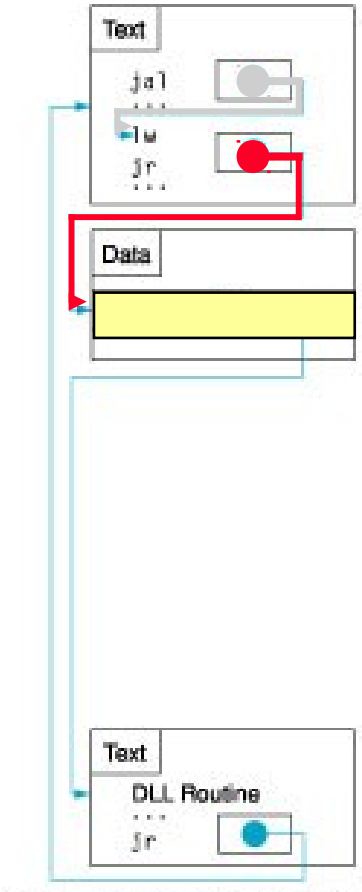
Passo 1: invocação da rotina não local



DLL:
invocações subseqüentes da rotina

Passo 1: invocação da rotina não local

Passo 2: interceptação pela rotina auxiliar

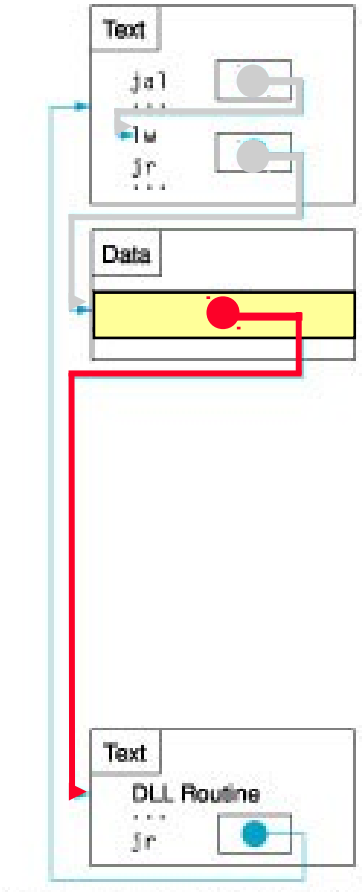


DLL:
invocações subseqüentes da rotina

Passo 1: invocação da rotina não local

Passo 2: interceptação pela rotina auxiliar

Passo 3: desvio para a rotina carregada



DLL:
invocações subsequentes da rotina

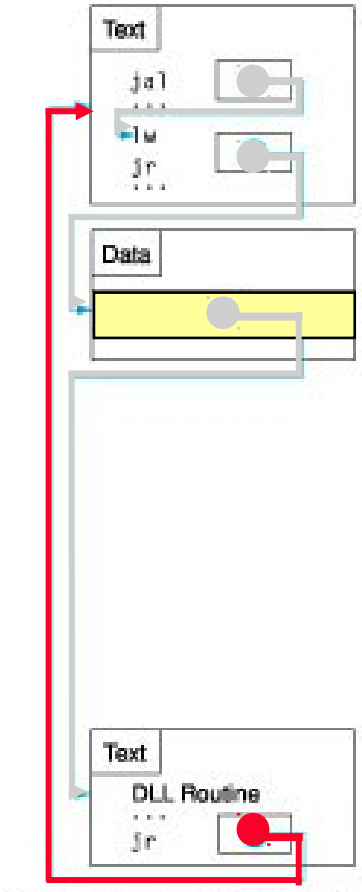
Passo 1: invocação da rotina não local

Passo 2: interceptação pela rotina auxiliar

Passo 3: desvio para a rotina carregada

Passo 4: retorno da rotina executada

Nota: instrução de desvio indireto (jr) permite o suporte a DLLs



DLL: avaliação

- **Espaço adicional em arquivo e memória (-)**
  - **Para armazenar informação extra**
    - » **Necessária à ligação dinâmica**
- **Degradação de desempenho (-)**
  - **Primeira invocação de rotina de biblioteca**
    - » **Tempo gasto para carga da rotina e sua ligação em tempo de execução**
  - **Invocações posteriores com *overhead* marginal**
    - » **2 desvios por invocação de rotina**

DLL: avaliação

- Upgrade automático de bibliotecas (+)
  - Em ambas as alternativas de implementação
- Uso eficiente do sistema de arquivos (+)
  - Em ambas as alternativas de implementação
- Uso eficiente de memória (+)
  - Na alternativa “Lazy Library Linkage”
    - » Uso difundido em sistemas de propósitos gerais
    - » Windows (.dll) e UNIX (.so)