

Dynamic Linked Library (.dll) e Shared Object (.so)

estudo

Lucas Skywalker

14 de agosto de 2015

1 Intro

As .dll ou .so são bibliotecas que contém rotinas já previamente compiladas. Para que o processo de reutilização dessas rotinas pelo seus programas seja mais ágil e menos custoso, pois enxergasse que serão reutilizadas muitas vezes pelos seus programas.

2 Estabelecendo ligação dinâmica pela primeira vez

1. De repente o sistema operacional se depara com uma rotina não local, ele gera um desvio para poder resolver o problema que apareceu para ele, de uma rotina contida em algum .dll ou .so, que será um `jal [LABEL1]` (o `jal` é porque ele deve retornar ao PC+4 depois dessa chamada para rotinas.)
2. A `[LABEL1]` é o local dentro do programa que irá ler a memória onde está guardada o endereço da sua rotina não local, isto é, ele irá executar `lw $s1, [LABEL2]`, carregando o endereço da rotina não local em um registrador (neste caso `$s1`), e logo após irá executar um `jr $s1` para desviar para o valor contido no endereço `[LABEL2]`. Como é a nossa primeira execução, o valor dentro de `[LABEL2]` não é a rotina que nós queremos ainda, ao invés disso ele irá ter o endereço do identificador da rotina, logo ele irá desviar para o identificador.
3. O código do identificador da rotina não local que o programa quer, será visto pelo montador como um `li $s0, I` (`li` = load immediate), logo após carregar `[LC]` ele desviará para o ligador/carregador (LC).
4. O LC irá carregar o código da rotina em memória e irá mapear o endereço do código para o endereço da memória `[LABEL2]`, aí então ele desviará para a rotina não local e a rotina será executada.
5. Executa a rotina não local e retorna para o programa.

Layout de Memória Antes e Depois da Ligação
dinâmica de um programa.o com a rotina printf

programa.o

programa.o
+ printf.o

0040 0000

```
jal LABEL1 # observe que o sist. op. desvia
            # princípio para outro lugar, a fim
            # de resolver o problema da ligação
            # dinâmica. Patterson página 147 2.12
```

0040 0054

```
jr $ra
```

....

0040 0100
0040 0104

```
lw $s1, LABEL2 # como que ele vai resolver
jr $s1          # esse endereço? Ele busca na
                # memória
```

....

0040 1000

```
li $s0, I # o que tem na memória é um
j LC      # endereço, que indica o local
          # onde está a rotina de ligamento/
          # carregamento. assim na primeira
          # chamada para rotina de dll,
          # acontece esse tramite todo, mas
          # depois você verá que ele não
          # precisará mais fazer todo esse
          # caminho. (Patterson página 147)
```

....

0040 FF00

```
(Código das rotinas de
remap. do LC)
....
```

```
j LD # o sistema executa o jump direto para
      # a rotina dinâmica (pré compilada) para
      # poder executar agora alguma
      # funcionalidade.
```

1000 0000

(programa.data)

1000 0018

1000 0100

0x0040 1000

esse primeiro layout é como se ainda não
tivesse sido ligado o programa.o com a
rotina printf.o do banco dinâmico.

isto quer dizer que esse endereço em
memória, de 32 bits, contém o local no
layout de memória para onde jr \$s1 irá
desviar pela primeira vez.

0040 0000

```
jal LABEL1
```

0040 0054

```
jr $ra
```

....

0040 0100
0040 0104

```
lw $s1, LABEL2 # digamos que você tenha que
jr $s1          # destrinchar esse lw, como
                # você faria? com o mínimo
                # de instruções possíveis.
```

....

```
res.: lui $at, 0x1001
      lw $s1, 0x100($s1)
      0x10010000
      + 0x00000100
      -----
      = 0x10010100
```

0040 1000

```
li $s0, I
j LC
```

....

load immed é pseudo-instrução
e vai ser modificada para poder
gravar em \$s0 um endereço de 32
bits.

0040 FF00

```
(Código das rotinas de
remap. do LC)
....
```

```
j LD
```

0041 0000

```
(Código da rotina printf)
```

....

0040 03FC

```
jr $ra
```

1000 0000

(programa.data)

1000 001C

primeiro programa.data tem o valor
apontando para 0x00401000 que foi
carregado em \$s1 usando duas instr.
implícitas.

depois desse quadro,
realizando a primeira vez
que ocorre a ligação, não
vai mais acontecer todo esse
trâmite, o endereço aqui vai
mudar para o endereço da
rotina printf, pois agora
ela já está carregada na
memória, certo? não tem
porque recalcular o seu
endereço novamente, basta
modificar aqui em
programa.data em 0x10000100
=> 0x00410000;

ver página 147 do livro
organização de computadores
Patterson.

```
(printf.data)
....
```

1000 0100

1000 010C