Bibliotecas

As bibliotecas são ficheiros com código compilado passível de ser reutilizado na produção de programas. Se o código em biblioteca for incorporado no ficheiro objeto executável, designa-se por ligação estática. Se o código em biblioteca for carregado em memória, apenas quando for invocado pela aplicação, designa-se por ligação dinâmica.

	vantagens	desvantagens
ligação estática	• execução mais rápida	• os ficheiros objeto executáveis são mais longos • em caso de atualização é necessário gerar novamente a aplicação
ligação dinâmica	• ficheiros objeto executáveis menores • em caso de atualização da biblioteca não é necessário gerar novamente aplicação • permite que o mesmo código seja usado por várias aplicações em simultâneo	• o tempo de procura da biblioteca e a resolução de símbolos afeta o tempo de execução do programa

Exemplo

Tomemos como exemplos os módulos (stack e fifo) e programa de aplicação seguintes. A opções de programação utilizadas são escolhidas com o critério de exemplificar situações técnicas e não por utilidade do código, eficiência ou elegância.

stack.h

```
#ifndef STACK_H
#define STACK_H

void stack_push(int
value);

int stack_pop();

extern int *stack_pointer;

#endif
```

stack.c

```
#include "stack.h"

#define SIZE_ARRAY(a) (sizeof(a) / sizeof(a[0]))

static int buffer[10];

int * stack_pointer = buffer + SIZE_ARRAY(buffer);

void stack_push(int value) {
      *--stack_pointer = value;
}

int stack_pop() {
      return *stack_pointer++;
}
```

fifo.h

```
#ifndef FIFO_H
#define FIFO_H

void fifo_insert(int value);
int fifo_remove();
int fifo_count;
int fifo_size;
```

fifo.c

```
#include "fifo.h"

#define SIZE_ARRAY(a) (sizeof(a) / sizeof(a[0]))

static int buffer[10];
static int *fifo_put = buffer, *fifo_get = buffer;
int fifo_count;
```

```
#endif
```

```
const int fifo_size = SIZE_ARRAY(buffer);

void fifo_insert(int value) {
      fifo_count++;
      *fifo_put++ = value;
      if (fifo_put == buffer + SIZE_ARRAY(buffer))
            fifo_put = buffer;
}

int fifo_remove() {
      fifo_count--;
      int value = *fifo_get++;
      if (fifo_get == buffer + SIZE_ARRAY(buffer))
            fifo_get = buffer;
      return value;
}
```

Consideremos o seguinte programa de aplicação:

main.c

```
#include <stdio.h>
#include "stack.h"
#include "fifo.h"

int a = 3;
int b = 8;

int main() {
      stack_push(a);
      b = stack_pop();

      fifo_insert(33);
      fifo_remove();
}
```

Biblioteca estática ou arquivo

As bibliotecas são armazenadas em ficheiros com nomes da forma: **libxxx.so** ou **libxxx.a**. Os ficheiros terminados em **.a** contém bibliotecas de ligação estática e os terminadas em **.so** contém ficheiros de ligação dinâmica. A sequência **xxx** é diferenciadora e identifica a biblioteca.

Criação

```
$ ar cr libcontainer.a stack.o fifo.o
```

Quando um símbolo contido numa biblioteca é referido, todo o código incluído no módulo a que pertence é incluído no ficheiro objeto executável. Para evitar a inclusão de código não utilizado nos executáveis é comum separarem-se as função da biblioteca por vários ficheiro fonte.

Ligação com bibliotecas estática

A biblioteca é um ficheiro que funciona como repositório de ficheiros objeto relocalizáveis.

Usar uma biblioteca estática é equivalente à ligação de vários ficheiros objeto relocalizáveis.

```
$ gcc -static main.o -o main -L. -lcontainer
```

O código em biblioteca é copiado para o ficheiro objeto executável. Em caso de alteração do código

da biblioteca, por funcionalidade ou correção de erros, é necessário produzir no ficheiro objeto executável.

Algoritmo do *linker*

Os símbolos definidos em bibliotecas são ignorados se na altura do processamento da biblioteca ainda não existirem referências para eles.

Por essa razão as bibliotecas são colocadas na linha de comando depois dos ficheiros objeto.

Exercício

Inverter a ordem de colocação da biblioteca em relação ao objeto **`main.o`**.

```
$ gcc -static -L. -lcontainer  main.o -o main
```

Biblioteca de ligação dinâmica

Criação

Os módulos que constituem a biblioteca são compilados em separado com a opção **`-fpic`**. Esta opção dá indicação ao compilador para gerar código que execute independentemente do endereço de memória onde for carregado.

```
$ gcc -c -fpic stack.c fifo.c

$ gcc -shared -o libcontainer.so stack.o fifo.o
```

O ficheiro produzido (**`libcontainer.so`**) é designado por *shared object*. Contém a informação necessária para se processar o seu carregamento em memória e integração em aplicações.

Ligação com bibliotecas dinâmica

Geração do programa executável:

```
$ gcc main.c -o app -L. -lcontainer
```

O programa de aplicação não precisa ser compilado independente da posição (opção -fpic).
O *linker* dá preferência à versão dinâmica da biblioteca (**`libcontainer.so`**). Aplica ligação estática (usa **`libcontainer.a`**) se a opção **`-static`** for indicada ou se a versão dinâmica (**`libcontainer.so`**) não existir.
Na ligação dinâmica o *linker* incorpora no ficheiro objeto executável meios de referência aos objetos da biblioteca e realiza a operação de relocalização das referências – PLTs.

O código da biblioteca é carregado em memória no momento da execução do programa se ainda não tiver sido carregado na sequência da execução doutro programa.

Dependências

No ficheiro objeto executável ficam registados os nomes de versão (soname) das bibliotecas necessária para a sua execução. O utilitário **`ldd`** permite verificar estas dependências e se estão acessíveis para carregamento.

```
$ ldd main
        linux-vdso.so.1 (0x00007ffeebdb3000)
        libcontainer.so => not found
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f022412e000)
```

```
/lib64/ld-linux-x86-64.so.2 (0x00007f022451f000)
```

Quando o programa for a executar o *loader* procura as bibliotecas no sistema de ficheiros por esta ordem:

1. no *path* incluído no executável definido pela opção **`-rpath`**;

   Incluir um *path* de pesquisa no próprio executável:

   ```
   $ gcc main.o -o main -L. -lcontainer -Wl,-rpath,/home/ezequiel/lib
   ```

2. nas diretorias indicadas na variável de ambiente **`LD_LIBRARY_PATH`**;

   Definir a variável de ambiente **`LD_LIBRARY_PATH`**:

   ```
   $ export LD_LIBRARY_PATH=/usr/ezequiel/lib
   ```

3. na *cache* – ficheiro **`/etc/ld.so.cache`**.

   A *cache* é atualizada pelo utilitário **`ldconfig`**, que introduz os *paths* definidos na linha de comando, definidos nos ficheiros **`/etc/ld.so.conf.d/*.conf`** e as diretorias **`/lib`** e **`/usr/lib`**.

   - Visualizar **`/etc/ld.so.conf.d/*.conf`**

     ```
     $ cat  /etc/ld.so.conf.d/*.conf
     /usr/lib/x86_64-linux-gnu/libfakeroot
     # Multiarch support
     /usr/local/lib/i386-linux-gnu
     /lib/i386-linux-gnu
     /usr/lib/i386-linux-gnu
     /usr/local/lib/i686-linux-gnu
     /lib/i686-linux-gnu
     /usr/lib/i686-linux-gnu
     # libc default configuration
     /usr/local/lib
     # Multiarch support
     /usr/local/lib/x86_64-linux-gnu
     /lib/x86_64-linux-gnu
     /usr/lib/x86_64-linux-gnu
     ```

   - Atualizar a cache:

     ```
     $ sudo ldconfig /home/ezequiel/lib
     ```

   - Verificar se se encontra na cache:

     ```
     $ ldconfig -p | grep libcontainer.so
     ```

Carregamento da biblioteca no arranque da aplicação

O *loader* carrega o programa e verifica que existe uma secção **.interp** com a indicação do *linker* dinâmico:

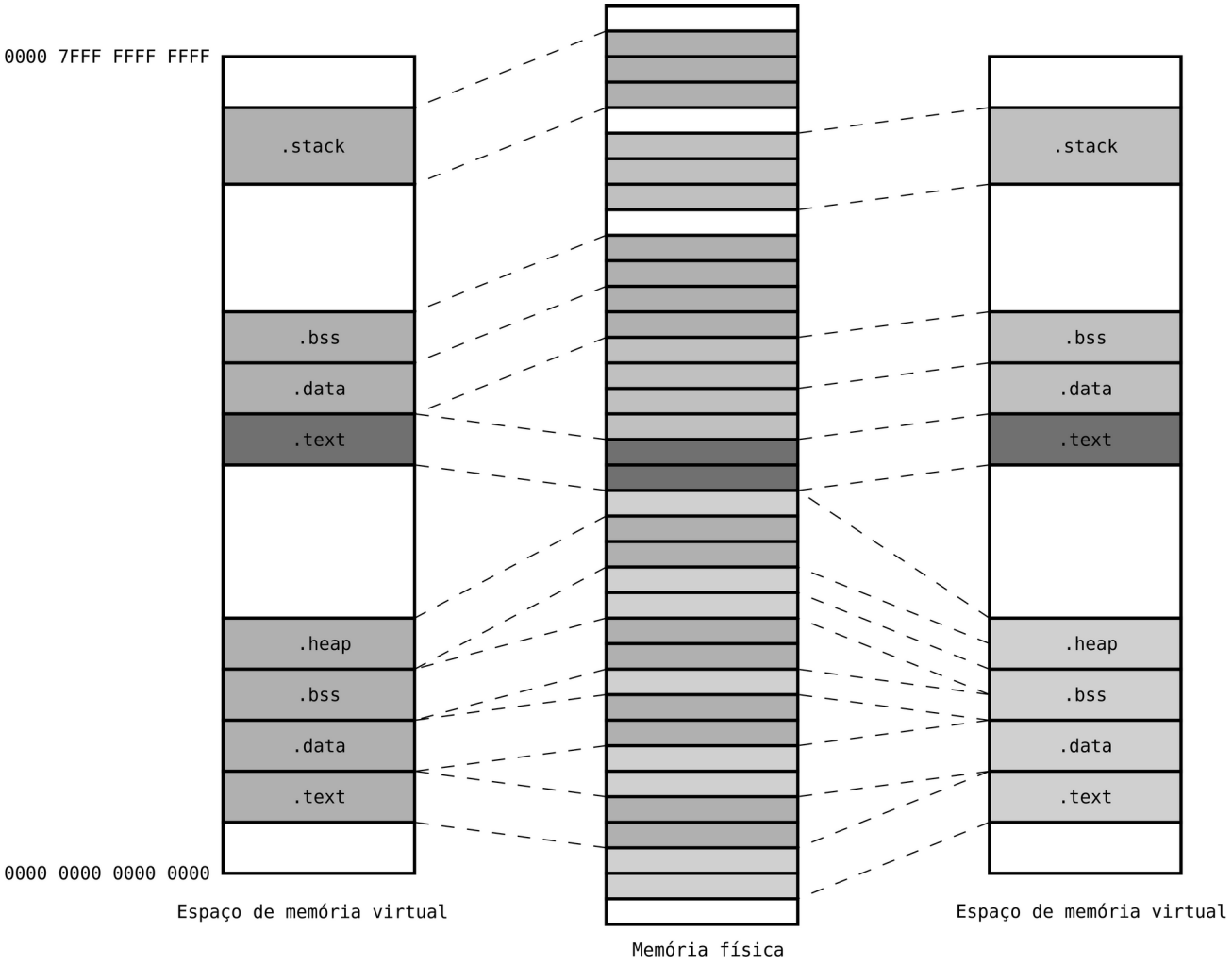
```
$ readelf -x .interp main

Hex dump of section '.interp':
  0x000002a8 2f6c6962 36342f6c 642d6c69 6e75782d /lib64/ld-linux-
  0x000002b8 7838362d 36342e73 6f2e3200          x86-64.so.2.          .2.
```

O *linker* dinâmico realiza as seguintes operações:

1. mapeia as secções de dados e de código, das bibliotecas dinâmicas que vão ser utilizadas, no espaço de memória do programa;
2. resolve as referências existentes no programa para dados e código das bibliotecas.

   (Esta operação não utiliza o processo das *relocations* porque isso implicaria alterações de código. É resolvido via GOT).



SONAME

nome de ligação (linker name)	libXXX.so	Corresponde geralmente a um *link*. É o nome que é usado para referenciar uma dada biblioteca na altura da geração do programa, sem definir a versão.
nome de versão (soname)	libXXX.so.N	O N indica a versão de especificação da biblioteca. Muda de versão sempre que a interface da biblioteca se torna incompatível com as anteriores. Corresponde normalmente a um *link*.
nome real	libXXX.so.N.M.R	Este é o ficheiro real onde se encontra o conteúdo da biblioteca.

N – interface incompatível com outras versões
M – interface compatível com versões anteriores
R – modificações internas

O objetivo deste esquema de nomes é facilitar as atualizações e lidar com várias versões.

Este esquema nem sempre é usado. Na distribuição Ubuntu 20.04 a libc usa um esquema diferente.

Procurar a libc instalada na máquina corrente

```
$ find / -name "libc.*"

...

$ cat  /usr/lib/x86_64-linux-gnu/libc.so

/* GNU ld script
   Use the shared library, but some functions are only in
   the static library, so try that secondarily.  */
OUTPUT_FORMAT(elf64-x86-64)
GROUP ( /lib/x86_64-linux-gnu/libc.so.6
/usr/lib/x86_64-linux-gnu/libc_nonshared.a  AS_NEEDED (
/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2 ) )
```

pkg-config

O utilitário **pkg-config** permite obter metainformação sobre bibliotecas instaladas. Essa informação inclui opções de compilação e caminhos para os locais de instalação. Essa informação é guardada em ficheiros, um por cada biblioteca, com a extensão **.pc**.

```
$ find /usr -name "pkgconfig"
/usr/local/lib/pkgconfig
/usr/lib/pkgconfig
/usr/lib/x86_64-linux-gnu/pkgconfig
/usr/share/pkgconfig
...

$ ls /usr/lib/x86_64-linux-gnu/pkgconfig/ -l
total 236
-rw-r--r-- 1 root root  275 fev  8  2019 dri.pc
-rw-r--r-- 1 root root  238 set 10 19:05 expat.pc
-rw-r--r-- 1 root root  506 abr  5  2018 fontconfig.pc
-rw-r--r-- 1 root root  326 abr 13  2018 freetype2.pc
-rw-r--r-- 1 root root  427 jan  8  2018 geany.pc
-rw-r--r-- 1 root root  314 jul 20  2017 geoclue-2.0.pc
-rw-r--r-- 1 root root  383 fev  8  2019 gl.pc
-rw-r--r-- 1 root root  223 mai 21  2016 glu.pc
-rw-r--r-- 1 root root  237 fev 28  2017 ice.pc
-rw-r--r-- 1 root root  253 fev 11  2018 jansson.pc
-rw-r--r-- 1 root root  239 jan  7  2018 json-c.pc
-rw-r--r-- 1 root root  301 jun 20 18:36 libcrypto.pc
-rw-r--r-- 1 root root 1858 set  6 06:27 libcurl.pc
...

$  cat  /usr/lib/x86_64-linux-gnu/pkgconfig/jansson.pc
prefix=/usr
exec_prefix=${prefix}
libdir=${prefix}/lib/x86_64-linux-gnu
includedir=${prefix}/include

Name: Jansson
Description: Library for encoding, decoding and manipulating JSON data
Version: 2.11
Libs: -L${libdir} -ljansson
Cflags: -I${includedir}


$ pkg-config jansson --libs

$ pkg-config jansson --cflags
```

Carregamento de bibliotecas em execução

Uma biblioteca de ligação dinâmica pode ser carregada em qualquer altura da execução da uma aplicação – não apenas no momento do arranque da aplicação.

API para carregamento de bibliotecas em execução:

```
#include <dfcn.h>
void *dlopen(const char *filename, int flags);
void *dlsym(void *handle, char *symbol);
int dlclose(void *handle);
const char *dlerror(void);
```

`dlopen` carrega a biblioteca indicada por **`filename`** e invoca o *linker* dinâmico para resolver referências a símbolos externos, definidos noutras bibliotecas ou no programa executável. Para que os símbolos definidos numa biblioteca fiquem disponíveis, essa biblioteca deve ter sido carregada com a *flag* **`RTLD_GLOBAL`**. Para que os símbolos definidos no programa executável fiquem disponíveis, esse executável deve ter sido criado com a opção **`-rdynamic`**.

`dlsym` recebe a referência para uma biblioteca previamente carregada com **`dlopen`** e o nome de um símbolo e retorna o endereço desse símbolo.

`dlclose` descarrega uma biblioteca previamente carregada com **`dlopen`**.

`dlerror` retorna uma *string* com a descrição do erro mais recente.

Exemplo 1

Fonte do programa executável:

```
#include <dlfcn.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
      if (argc != 2) {
            fprintf(stderr, "usage: %s libXXX.so\n", argv[0]);
            return -1;
      }

      void* handle = dlopen(argv[1], RTLD_LAZY);
      if (handle == NULL) {
            fprintf(stderr, "%s\n", dlerror());
            return -1;
      }

      int (*f)(void) = dlsym(handle, "lib_func");
      if (f == NULL) {
            fprintf(stderr, "Could not find lib_func: %s\n", dlerror());
            return -1;
      }

      printf("Calling lib_func\n");
      int ret = f();
      printf("lib_func returned %d\n", ret);

      if (dlclose(handle) != 0) {
            fprintf(stderr, "Could not close plugin: %s\n", dlerror());
            return -1;
      }
```

```
      return 0;
}
```

Para gerar o executável:

```
$ gcc main.c -ldl -o main
```

Fonte do programa em biblioteca:

```
#include <stdio.h>

int lib_func() {
      printf("Executing lib_func\n");
      return 77;
}
```

Geração da biblioteca de ligação dinâmica:

```
$ gcc -fpic lib_func.c -shared -o lib_func.so
```

1ª experiência

```
$ ./main
usage: ./main libXXX.so
```

Não foi indicado ficheiro da biblioteca.

2ª experiência

```
$ ./main lib_func.so
lib_func.so: cannot open shared object file: No such file or directory
```

O ficheiro da biblioteca não foi encontrado porque a diretoria corrente não se encontra nos caminhos de procura de bibliotecas.

3ª experência

```
$ export LD_LIBRAY_PATH=.
$ ./main lib_func.so
Calling lib_func
Executing lib_func
lib_func returned 77
```

Exemplo 2

Capacitar o programa de simulação de fila de espera para aceitar a incorporação de novos comandos através da ligação dinâmica em execução (*plugin*).
Os comandos do programa são agrupados numa lista ligada em que a informação relativa a cada comando é composta pelo ponteiro para a função que executa o comando, a descrição textual do comando e a letra identificadora.
A incorporação de novo comando é realizada pela função **`commando_new`** – linhas 36 a 58. Esta função recebe o nome do ficheiro contendo o *shared object* e extrai através da função **`dlsym`** os ponteiros para os elementos do novo comando.
A partir do momento em que estes elementos são inseridos na lista de comandos, o novo comando passa a estar disponível, a par dos comandos originais.

```
  1 typedef struct command {
```

```
2       void (*f) (char *);
3       char c;
4       char *desc;
5       struct command *next;
6 } Command;
7
8 static Command *commands = NULL;
9
10 void command_insert(char c, char *desc, void (*f)(char *)) {
11      Command *new_command = malloc(sizeof (Command));
12      new_command->c = c;
13      new_command->desc = strdup(desc);
14      new_command->f = f;
15      new_command->next = commands;
16      commands = new_command;
17 }
18
19 void command_execute(char c, char *param) {
20      for (Command *p = commands; p != NULL; p = p->next)
21            if (p->c == c) {
22                  p->f(param);
23                  return;
24            }
25 }
26
27 void command_list(char *unused) {
28      for (Command *p = commands; p != NULL; p = p->next)
29            printf("%c%s\n", p->c, p->desc);
30 }
31
32 #include <dlfcn.h>
33
34 static void *handle;
35
36 static void command_new(char *lib) {
37      handle = dlopen(lib, RTLD_LAZY);
38      if (handle == NULL) {
39            fprintf(stderr, "%s\n", dlerror());
40            return;
41      }
42      void (*f)(char *) = dlsym(handle, "command_function");
43      if (f == NULL) {
44            fprintf(stderr, "%s\n", dlerror());
45            return;
46      }
47      char *c = dlsym(handle, "command_letter");
48      if (c == NULL) {
49            fprintf(stderr, "%s\n", dlerror());
50            return;
51      }
52      char **desc = dlsym(handle, "command_description");
53      if (desc == NULL) {
54            fprintf(stderr, "%s\n", dlerror());
55            return;
56      }
57      command_insert(*c, *desc, f);
58 }
59
60 int main() {
61      char line[100];
62      command_insert('s', "\t - Sair", leave_program);
63      command_insert('h', "\t - Listar comandos existentes", command_list);
```

```
64        command_insert('c', "\t - Incorporar novo comando", command_new);
65        command_insert('d', " <nome> - Desistencia de utente", user_remove);
66        command_insert('l', "\t - Listar fila de espera", user_print);
67        command_insert('a', "\t - Atender utente", user_answer);
68        command_insert('n', " <nome> - Chegada de novo utente", user_insert);
69
70        while (1) {
71              putchar('>');
72              fgets(line, sizeof line, stdin);
73              char *command = strtok(line, " \n");
74              char *name = strtok(NULL, " \n");
75              if (command != NULL)
76                    command_execute(*command, name);
77        }
    }
```

O executável é gerado sob o controlo do seguinte *makefile*:

```
CFLAGS = -g -Wall

wqueue: wqueue.o readline.o
      gcc wqueue.o readline.o -ldl -o wqueue -rdynamic

wqueue.o: wqueue.c
      gcc -c $(CFLAGS) wqueue.c

clean:
      rm -rf *.o wqueue
```

A criação de um novo comando consiste em gerar um *shared object* contendo os elementos do novo comando: a função que executa o comando, a descrição textual e a letra identificadora.
Em seguida exemplifica-se a criação de um comando para vazar fila de espera.

```
 1 #include <stdlib.h>
 2 #include "user.h"
 3
 4 extern User queue;
 5
 6 void command_function(char *name) {
 7       User *next;
 8       for (User *p = queue.next; p != &queue; p = next) {
 9             next = p->next;
10             p->prev->next = p->next;
11             p->next->prev = p->prev;
12             char *aux = p->name;
13             free(p);
14             free(aux);
15       }
16 }
17
18 char command_letter = 'k';
19
20 char *command_description
21               = "\t - Eliminar todos os utentes da fila de espera";
```

Os elementos do comando são obrigatoriamente designados pelos símbolos **`command_function`**, **`command_letter`** e **`command_description`**, pois serão estes os símbolos utilizados por parte do programa executável, ao invocar a função **`dlsym`** – linhas 40, 45 e

50.

O *shared object* é gerado sob o controlo do seguinte *makefile*:

```
libleak.so: leak.o
      gcc -shared leak.o -o libleak.so

leak.o: leak.c
      gcc -c -fpic leak.c
```

1ª experiência

```
$ .wqueue
>c libleak.so
libleak.so: cannot open shared object file: No such file or directory
>
```

O ficheiro da biblioteca não foi encontrado porque a diretoria corrente não se encontra nos caminhos de procura de bibliotecas. Para o incluir, recorre-se à variável de ambiente **LD_LIBRARY_PATH**.

```
$ export LD_LIBRARY_PATH=.
```

2ª experiência

Para efeito desta experiência, o executável **wqueue** foi gerado sem a opção **-rdynamic.**

```
$ ./wqueue
>c libleak.so
./libleak.so: undefined symbol: queue
>
```

Este erro deve-se ao facto de o programa executável, por omissão, não exportar símbolos para ligação dinâmica com os *shared objects*. A referência indefinida ao símbolo externo **queue,** existente em **libleak.so**, não poder ser resolvida pelo *linker* dinâmico.

```
$ readelf --dyn-syms libleak.so
Symbol table '.dynsym' contains 10 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
       ...
     4: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND queue
       ...
```

O símbolo **queue**, definido no programa executável, só fica disponível para ligação dinâmica, se o executável for gerado com a opção **-rdynamic** – linha 4 do *makefile*. Nessa altura todos os símbolos globais são incluidos na tabela de símbolos de ligação dinâmica.

```
$ readelf --dyn-syms wqueue
Symbol table '.dynsym' contains 40 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
       ...
    27: 0000000000004010    24 OBJECT  GLOBAL DEFAULT   25 queue
       ...
```

3ª experiência

Depois de o programa executável ser produzido com a opção **-rsymbol** a ligação dinâmica do símbolo **queue** sucede bem e o código acabado de carregar pode ser utilizado.

```
$ ./wqueue
>c libleak.so
>h
k      - Eliminar todos os utentes da fila de espera
n <nome> - Chegada de novo utente
a      - Atender utente
l      - Listar fila de espera
d <nome> - Desistencia de utente
c      - Incorporar novo comando
h      - Listar comandos existentes
s      - Sair
>
```