Programación de Sistemas CCPG1008

Federico Domínguez, PhD.

Unidad 4 – Sesión 1: Librerías estáticas y dinámicas

Contenidos

Librerías en C

Librerías estáticas

Librerías dinámicas

Librerías con make

Librerías en C

Una librería es una colección de implementaciones de algoritmos o funciones diseñadas a ser usadas mediante una interfaz predefinida.

Pueden estar disponibles en código fuente u *object files*.

No están diseñadas para ser ejecutadas por si solas, tienen que ser accedidas y usadas en algún programa.

Sirven para:

- facilitar el reuso de código,
- organizar algoritmos, funciones y comportamientos en archivos distribuibles,
- compartir código abierto o propietario y
- optimizar el uso de recursos computacionales.

En C y C++ se las "*invoca*" con la directiva de preprocesador `#include`.

- Equivalente (aunque no igual) a `import` en Java o Python

Librerías en C y el uso de la directiva **#include**

La directiva de preprocesador **#include** es típicamente asociada con el uso de librerías.

El preprocesador al encontrar la línea:

```
#include “file.h”
```

Reemplaza la línea con el contenido completo de *file.h*. Existen dos formas de expresar la directiva:

1. `#include “file.h”`
2. `#include <file.h>`

Con < > *file.h* es buscado primero en directorios del sistema predefinidos donde se encuentran librerías de C o C++. Luego busca en directorios especificados con la opción `-I` de *gcc*.

Con “ ” *file.h* es buscado primero en el directorio del proyecto, luego en directorios especificados al compilar usando la opción `-iquote`, y finalmente en los directorios usando < >

Sin la directiva **#include**, la gestión de librerías y mantenimiento de código sería onerosa.

```c
int add(int a, int b)
{
        return a + b;
}
```

add.c

```c
int add(int, int);

int triple(int x)
{
    return add(x, add(x, x));
}
```

archivo1.c

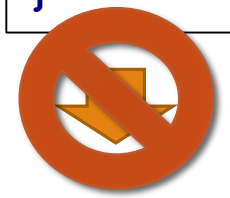
```c
int add(int, int);

int triple_neg(int x)
{
    return add(-1*x, add(x, x));
}
```

archivo2.c

Sin la directiva **#include**, la gestión de librerías y mantenimiento de código sería onerosa.

```c
float add(float a, float b)
{
        return a + b;
}
```

add.c



```c
int add(int, int);

int triple(int x)
{
    return add(x, add(x, x));
}
```

archivo1.c

```c
int add(int, int);

int triple_neg(int x)
{
    return add(-1*x, add(x, x));
}
```

archivo2.c

Sin la directiva **#include**, la gestión de librerías y mantenimiento de código sería onerosa.

```c
float add(float a, float b)
{
        return a + b;
}
```

add.c

```c
float add(float, float);

int triple(int x)
{
    return add(x, add(x, x));
}
```

archivo1.c

```c
float add(float, float);

int triple_neg(int x)
{
    return add(-1*x, add(x, x));
}
```

archivo2.c

Al incluir los prototipos de funciones en archivos de **cabecera .h** facilitamos el problema de gestión de librerías de código.

Los archivos de cabecera son conocidos como *header files.*

```
float add(float, float);
```

add.h

```
#include "add.h"

int triple(int x)
{
    return add(x, add(x,x));
}
```

archivo1.c

```
#include "add.h"

int triple_neg(int x)
{
    return add(-1*x, add(x,x));
}
```

archivo2.c

“Header guards” son comúnmente usadas para evitar incluir dos veces un archivo de cabecera.

Típica implementación de **add.h**:

```
/* add.h */
#ifndef ADD_H_
#define ADD_H_

float add(float, float);

#endif
```

Librerías estáticas

El lenguaje C permite el uso de librerías estáticas para optimizar el reuso y redistribución de código y funcionalidades.

Una librería estática en C es un empaquetamiento de funciones compiladas en archivos con extensión .a.

Los archivos de librerías son *object files* con un formato que permite *linking* especifico de funciones individuales.

Por ejemplo si deseo compilar un archivo `main.c` el cuál usa algunas funciones en `libm.a` y `libc.a` podría hacerlo así:

```
gcc main.c /usr/lib/libm.a /usr/lib/libc.a
```

La librería `libc.a` es la librería estándar de C (`printf, scanf, atoi, strcpy`, …), es incluida por defecto durante el proceso de *linking* y no es necesario especificarla.

El formato ***archive*** (con extensión .a) es una forma de empaquetamiento de *object files*.

Para archivar funciones y crear nuestra propia librería estática:

```
gcc –c addvec.c multvec.c
ar rcs libvector.a addvec.o multvec.o
```

```c
void addvec(int *x, int *y,
            int *z, int n)
{
    int i;

    for (i = 0; i < n; i++)
        z[i] = x[i] + y[i];
}
```

addvec.c

```c
void multvec(int *x, int *y,
             int *z, int n)
{
    int i;

    for (i = 0; i < n; i++)
        z[i] = x[i] * y[i];
}
```

multvec.c

Para usar cualquier librería es necesario declarar los prototipos en archivos .h

```
gcc –c main2.c

gcc –static –o prog main2.o ./libvector.a
```

```
1   /* main2.c */
2   #include <stdio.h>
3   #include "vector.h"
4
5   int x[2] = {1, 2};
6   int y[2] = {3, 4};
7   int z[2];
8
9   int main()
10  {
11      addvec(x, y, z, 2);
12      printf("z = [%d %d]\n", z[0], z[1]);
13      return 0;
14  }
```

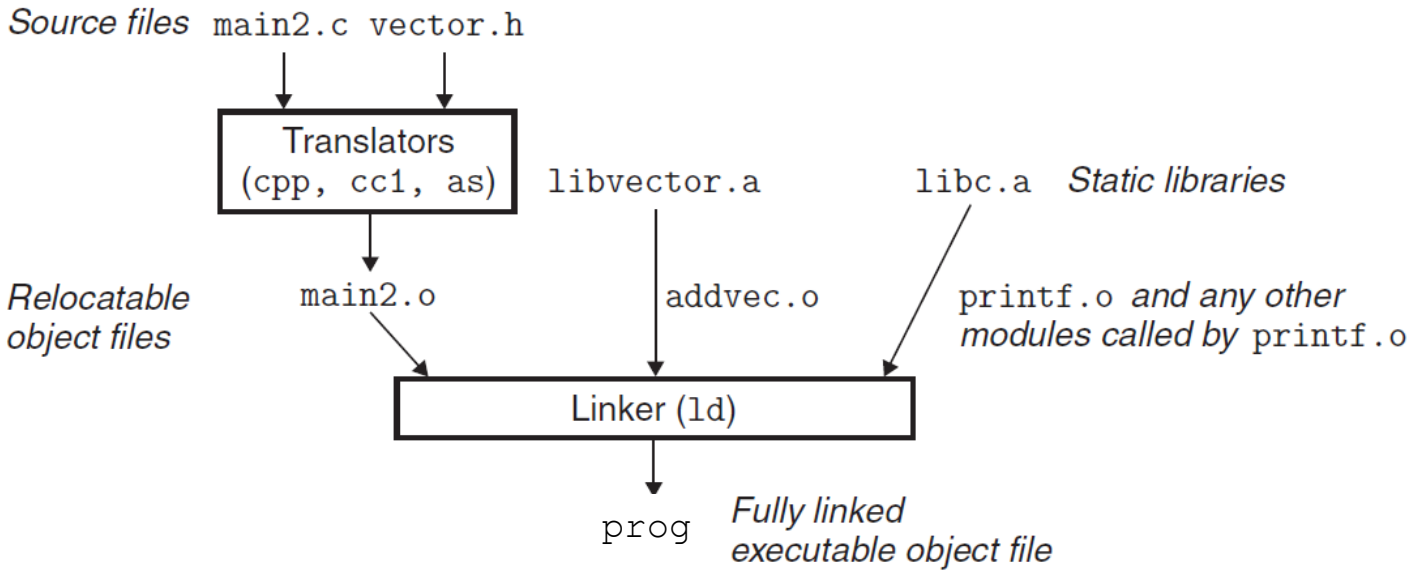
Durante *linking*, tan solo las funciones usadas de una librería estática son incluidas en el ejecutable.

O también

```
gcc –c main2.c
```

Equivalente: `gcc –static –o prog main2.o ./libvector.a`

```
gcc –static –o prog main2.o –L. -lvector
```



Al usar varias librerías, desafortunadamente el orden de especificación en el comando **gcc** sí importa.

Es necesario especificar el archivo que usa las funciones antes que la librería que las implementa.

La regla general es posicionar siempre las librerías al final de la línea de comandos.

```
unix> gcc -static ./libvector.a main2.c
/tmp/cc9XH6Rp.o: In function ‘main':
/tmp/cc9XH6Rp.o(.text+0x18): undefined reference to ‘addvec'
```

Librerías estáticas con **make**

Demostración

Librerías dinámicas

Las librerías estáticas tienen dos desventajas:

- No facilitan la actualización de software, es necesario recompilar si aparece una nueva versión de una librería.
- No son tan eficientes, por ejemplo si existen 50 programas en memoria que usan la función `printf`, la función estaría repetida 50 veces en disco duro y memoria.

Las librerías dinámicas son una invención relativamente moderna que resuelven este problema.

Son también conocidas como *shared libraries* y tienen la extensión `.so`.

Son equivalentes a las *dll* en Microsoft Windows.

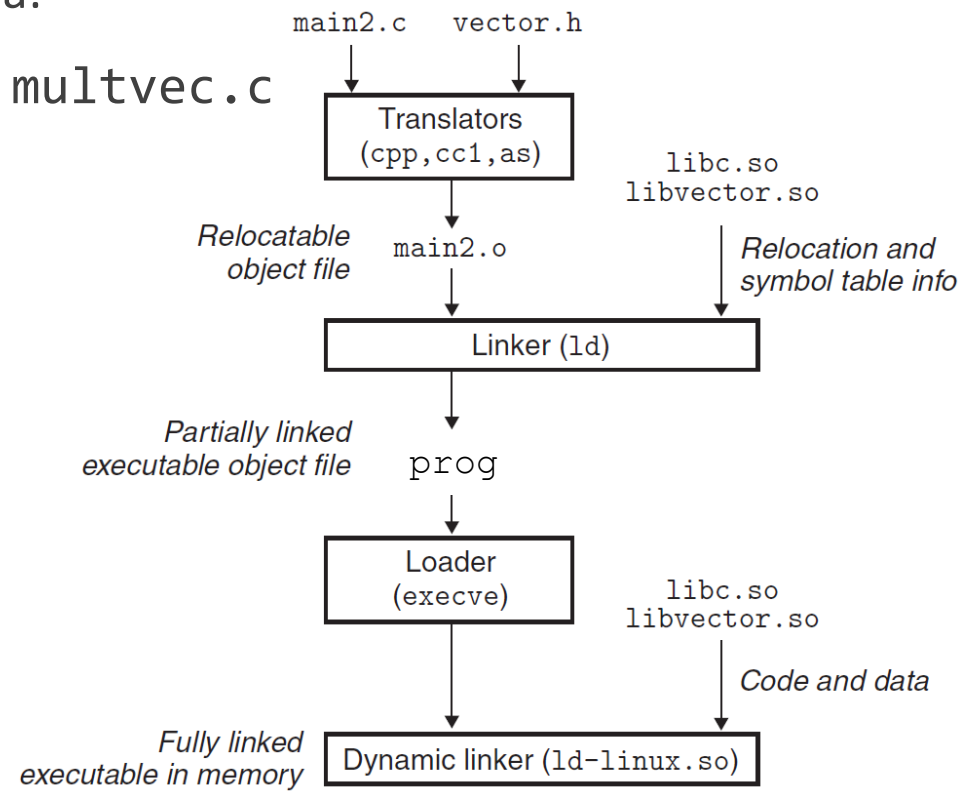
Las librerías dinámicas son cargadas en **tiempo de carga** en lugar de compilación.

Una librería dinámica puede crearse de la siguiente manera:

```
gcc -shared -fPIC -o libvector.so addvec.c multvec.c
```

Y puede ser usada:

```
gcc -o prog main2.c ./libvector.so
```



Las librerías dinámicas también pueden ser cargadas en **tiempo de ejecución**.

Los sistemas Linux proveen una interface al *dynamic linker* mediante `dlfcn.h`:

```
#include <dlfcn.h>

void *dlopen(const char *filename, int flag);
```

Returns: ptr to handle if OK, NULL on error

```
#include <dlfcn.h>

void *dlsym(void *handle, char *symbol);
```

Returns: ptr to symbol if OK, NULL on error

Librerías dinámicas con make

Demostración

Para la próxima clase

Libro Computer Systems, Bryant y O'Hallaron. Secciones 7.6.2-3, 7.9 – 7.11

Práctica: Creación de librerías en C